Natural Language Processing for Foreign Language Learning

Jacob Kausler
Ouachita Baptist University

SENIOR THESIS APPROVAL

This Honors thesis entitled

"Natural Language Processing for Foreign Language Learning"

written by

Jacob Kausler

and submitted in partial fulfillment of
the requirements for completion of
the Carl Goodson Honors Program
meets the criteria for acceptance
and has been approved by the undersigned readers.

Dr. Jeff Matocha, thesis director

Dr. Joey Dodson, second reader

Dr. Darin Buscher, third reader

)r. Barbara Pemberton, Honors Program director

4/20/2015

Natural Language Generation for Foreign Language Learning

Second Draft

Jacob Kausler

February 2015

For my Pawpaw,
You taught me that the purpose of an education was not to open new doors, but to keep old doors open.

Contents

List of Figures

Abstract

This research presents novel algorithms which generate sentences in a natural language, using natural language generation techniques. The purpose of the algorithms is to benefit foreign language learning. As far as we can tell, ours is the first such research being done in the field. In creating the algorithms, we also developed a piece of software to showcase the work and allow testing by users. The main algorithm begins by generating sentence models by using one of two methods, namely modeled sentence generation and semantic sentence generation. Each of these have benefits and drawbacks, which the user must take into consideration when generating sentences. When the models are generated, they are filled in word by word using a conjugation algorithm. The completed sentences are then returned to the user and may then be translated. There is still much work to do before we will be satisfied with the algorithms, but our research shows that it is possible to use natural language generation techniques to benefit foreign language learning. '

Chapter 1

Introduction

The study of another language opens one's mind to new cultures and ways of life. Ancient languages can widen a student's perspective even further. Not only do they take the reader around the world, but they also take them back in time. There are great benefits to studying these languages.

One of the best ways to become familiar with a language is through practice. However, in the case of ancient languages, it is nearly impossible to find someone who is able and willing to communicate with the student. This l'eaves one option: he or she must find texts to translate in order to keep up their knowledge of the language. The problem with this method is that it can be burdensome to find a piece of literature in the target language every time he or she wants to practice. For some rarer languages, it can be a near impossible task. For example, in Ugaritic, the only existing manuscripts are those from the ancient city of Ugarit, dating from the fourteenth to twelfth century[28].

We propose a new method for translation practice,*natural language generation for foreign language learning*. By utilizing natural language generation techniques, we have constructed novel software with which students of foreign languages can practice translating sentences that are not found in any corpus. As far as we can tell, ours is the first such utilization of natural language generation to benefit foreign language learning. The sentences are constructed based on existing literature of the desired language, using its vocabulary, syntax, and style. The result is an algorithm that can construct sentences found nowhere in the existing literature, but that are of a related style such that they can be used by students to maintain their language skills.

Tbis chapter provides a brief history of natural language processing and generation. It also gives an overview of work related to the nature of this product. Much of the background information in this chapter comes from Jurafsky's *Speech and Language Processing*[13].

1.1 Background Information

The nature of this project is rooted deeply in the existing field of computer science called natural language processing (NLP). Therefore, it is important to understand previous work that has been done in this field related to foreign language generation. To the best of our knowledge, the use of natural language generation to assist students in foreign language has never been used. There are, however, similar fields that have influenced this project.

1.1.1 Natural Language Processing

The field of natural language processing is one of the oldest fields of study in computer science. It grew out of the work of several pioneers of computers, including Turing, Kleene, and Chomsky. When the idea of automatons, theoretical machines that could model various processes, grew out of Turing's work, people almost immediately began applying it to linguistic study. For example, in 1943 McCulloch and Pitts used automaton to study and model the behavior of propositional logic. Kleene developed both the ideas of finite automata and regular expressions as ways to model regular languages. Chomsky went further, and developed ways to represent deeper levels of languages, including context-free languages. There were various other branches of NLP developed during this period, including speech processing and optical recognition; however, they are unrelated to this project and will not be discussed. The interested reader should look into the introductory chapters of NLP textbooks for further information.

After Chomsky, the field divided into two main branches. First were those who used *symbolic methods* of research. These were researchers who, like Chomsky and others, developed automata and other theoretical machines to complete tasks. Those who worked in formal language theory developed processes that worked mainly with parsing problems, which parsed and labeled words based on their parts of speech or some other factor. Others, working in artificial intelligence (AI), created processes to handle reasoning and logic-based problems. These became foundational in question-answering systems and similar work.

The second group used *stochastic methods* of research. These were people who applied probability-based solutions to their problems. Emerging mainly from a statistical background, this area gave rise to several important methods for probability used in NLP today, including the application of the Bayesian method to NLP problems. One of the key problems addressed by this group involved sequence probabilities, and was used in both optical character recognition (OCR) and authorship attribution.

In the 1970s, four main areas of research emerged. Because of its probabilistic nature, the stochastic researchers began to explore the problem of *speech recognition*. The work done in this field augmented other areas of research, bringing the hidden Markov model (HMM) and noisy channel algorithms to light.

Another area dealt with *logic-based programming* in more detail. Having already been given heavy attention by researchers of AI, this field already had a solid base going into this new era. One of the major breakthroughs during this new decade was the development of the metamorphosis grammar, the predecessor of the definite clause grammar, which played an important role in the development of feature structure unification, an important part of this project.

Third was the ascent of *natural language understanding*. This field was related to that of logic-based programming; however, instead of simply realizing the logic behind statements, natural language understanding allowed programmers to perform actions based on natural language commands. This field saw the development of the first major grammar of English, and was also an important reason predicate logic began to be used in semantic representations.

Finally, this decade saw the beginning of *discourse modeling*. Researchers in this field attempted to create algorithms that could understand discourse structure and focus, as well as the related problems of reference resolution and speech act theory in discourse.

The next major research shift happened in the late 1990s. Up to this time, the various camps of NLP had slowly begun to blend together. This process was accelerated at the end of the millennium (as is continuing to occur) as researchers grew more connected. New problems emerged, revolving around the Internet and how information was to be retrieved effectively. As computers sped up and became more powerful, researchers were able to utilize new techniques that would have been impossibly slow on old machines.

In the new millennium, much of the research done in NLP involved machine learning. Researchers were becoming interested in discovering how much a computer could learn and relay information to the user. Because machine learning requires large portions of training material to effectively create algorithms, and because of the new ease in sharing information, the past fifteen years have seen a tremendous increase in the amount of training material, both labeled and unlabeled, available to researchers.

1.1.2 Natural Language Generation

Throughout the decades of NLP research, other research areas have arisen. One of these is natural language generation (NLG). This field is the main focus of our research, and therefore we will give a brief history of the field, separate from the history we have given of NLP above. While some work was done in the early years of NLP, it was very primitive and impacted the overall field very little. It was not until the 1960s that meaningful work began to be produced.

Early work in this field involved machine translation. Researchers were interested in finding the best ways to translate between two languages and produce coherent, grammatically correct sentences. The 1960s saw the first breakthrough of NLG in this field, as researchers began to use grammars to produce well-formed sentences.

Most of the work in NLG of the past fifty years consisted of the construction of text representing non-linguistic data in a readable form. Therefore, most research was done in order to develop better methods of relating this data to some user. In the early period of NLG, researchers equated the field to natural language understanding, thinking that by making the system understand the data it would be able to better output the data.

However, in the early 1980s, many researchers began to realize that a good NLG system could not easily be constructed out of an understanding system. Therefore, researchers began to research the field independently. Work was done on large systems at first, but most of these were scrapped as researchers noted that until they could develop small, focused systems, they would not be able to develop systems to work with wide knowledge bases.

Researchers in the field soon realized they were right about NLG being separate from understanding tasks. In order to relay information to the user after understanding the information, researchers knew that much work would have to be done to create both working algorithms, and ones that could do the work efficiently. However, after separating the branches, researchers noticed that by giving the system only a small understanding of the knowledge base, good text could still be generated. Therefore, most of the work of the past twenty years has consisted of creating accurate, well-formed text, as opposed to making the systems understand the knowledge behind the text.

1.1.3 Related Work

As far as we can tell, our work is a new area of research in natural language processing. There are related systems in NLG, but none generate sentences in a foreign language for the purpose of practice translation.

Currently, the most well-known software for foreign language learning is Rosetta Stone[2]. This program attempts to teach students foreign languages through a process known as spaced repetition. In this process, new vocabulary or grammatical concepts are shown to the student, and, depending on whether it was answered correctly or incorrectly, an algorithmic method determines how long the program must wait before showing the question again. The more often a question is answered correctly, the less often it is shown. Some flashcard programs also use spaced repetition, including Anki[8], Mnemosyne[1], and SuperMemo[3]. All of these, however, require programmers or users to manually enter the data (or import it from another source) for the questions, and do not rely on NLG techniques.

Natural language generation has played a role in machine translation. In this field, a hierarchy of methods has been theorized. At the base is direct translation, where individual words are translated one by one. This is extremely rudimentary and lacks any need for structured generation. The second level is syntactic translation. At this level, phrases are translated into the target language, accounting for differences in grammatical rules. This is currently the most widespread model of translation, and relies on NLG to generate a grammatically correct translated document. The third and fourth levels of translation use semantic meanings to translate, first translating into an abstract language that represents the meaning of the sentence, then translating into the target language using this meaning. Few systems have implemented this type of translation, and those that have are small, domain specific systems. This is currently an important field of research.

Aside from its use in translation, NLG is mainly used in three key areas: question-answering systems, conversational agents, and data summarization. These three are parts of overlapping fields of research, and all have seen important applications during their development. *Question-answering systems* are just what they seem; a user queries the system and the system replies with the answer. User queries can take several forms, from simple keyword systems to complex natural language statements. The system takes the query and analyzes its knowledge base to return an answer in natural language.

The second field of research involves *conversational agents*. Conversational agents are systems with which the user can converse. They can take many forms, from simple, one-line joke generators (currently a popular use of NLG) to complex tutorial systems. These agents have several parallels with question-answering systems, in that sometimes they must answer user questions. However, question-answering systems are typically reliant on a single query, while conversational agents must be able to analyze the surrounding context to determine the nature of a question. Neither are all conversational agents purely interested in answering questions correctly, or even at all. Some agents, especially ones which are interested only in researching well-formed generation methods, create systems that, when asked something by the user, turn the asker's words around, dodging the question.

The final research area is *data summarization*. This is probably the most popular area of research in NLG. Researchers develop tools for summarizing and describing data. Because question-answering systems rely on representing data to the user, they will often use methods developed in this field. However, data summarization is much broader than systems designed to answer user queries. These systems are not only used to represent data sets in natural language, but also to summarize existing documents, finding the key points and structures of the document and providing a brief abstract.

As can be seen, the areas of research in natural language generation are very broad. However, none have yet attempted to use NLG methods in foreign language learning. Therefore, while there are many algorithms with which to base our new area of research, the field itself is a novel contribution to natural language generation, as well as to its parent field, natural language processing.

1.2 What is to Come

The next chapter describes the layout of the graphical user interface and gives a description of the overall sentence generation algorithm. We dedicate two chapters to describing the two model generation methods, modeled sentence generation and semantic sentence generation; we explain the process of filling in sentence models with actual words in Chapter Five. Then, Chapter Six gives a description of the system as we implemented it, including our choice of corpora and the development process. We conclude with a description of the remaining problems and future work planned for the project.

In addition to the main text of this work, we include three appendices. The first is the Geting Started Guide document, which is to be provided with the program when distributed to users. Second is the Software Requirements Specification document, which was created before development started to outline the requirements of the system. Finally, we include a test suite, containing example sentences generated by each method.

A CD containing the software described is included with this text. In order to run this software, the user must have Python 2.7 installed on their system. For installation instructions, see Appendix A.

Chapter 2

User Interaction and Top-Level Design

We start the discussion of our program design by describing the top-level functionalities of the system. First, we discuss the reasoning behind the graphical user interface (GUI) layout. Then we examine the main algorithm, discussing the process it follows in calling sub-methods that, together, generate a sentence that is returned to the user.

2.1 Layout of the GUI

Despite the thousands of lines of code required to make the system run, most users will not think beyond the layout that makes up the main window. Because of this, we designed the GUI such that users of any level of technical skill can take full advantage of it. Figure 2.1 shows a screenshot of the program's GUI.

The first thing the user will notice are two large text boxes in the center of the screen. The top box, a read-only field labeled "Sentence", will hold one of the sentences generated by the program. The second box, labeled "Translation", will hold the user's (optional) translation of that sentence. We wanted to make the two boxes as self-explanatory as possible, and hope that the labels make their roles clear.

After the text boxes, the user's attention will be directed to the bottom of the window, where there is an array of buttons and sliders. Each of these are labeled and, with the help of frames, guide the user in their meaning. The "Previous" and "Next" buttons navigate to backward and forward, respectively, in the list of generated sentences. Upon reaching the beginning or end of the list, the system cycles back to the other end for a continuous loop.

A feature that we decided would be helpful for the user is to have the system automatically save translations when navigating forward and backward. Therefore, any time the "Previous" or "Next" buttons are pressed, the system stores the user translations in memory, with the same key as the sentence. Then, whenever the displayed sentence is changed, the system must simply lookup the key of the current sentence in the saved translations and output them into the bottom box. The user need not worry about having to save their work as they go.

This feature is limited only to the current session, however. Should the user close the

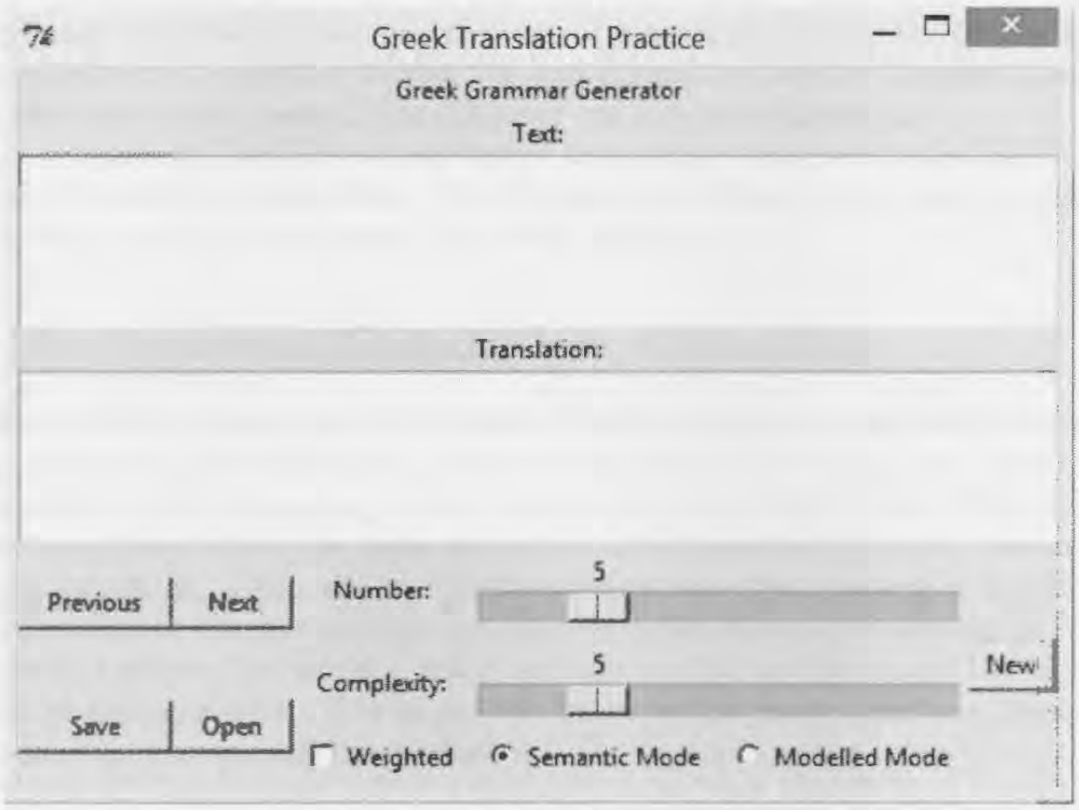

Figure 2.1: GUI of the program

software or choose to generate new sentences, the current list will be lost. Therefore, we have also included two other buttons: one to allow the user to save their current session, and another to allow the user to open a previously saved session.

Because there are several times when the user could potentially lose their work, we have chosen to include a notification feature that warns the user if his or her current session will be erased by their action. It gives them the opportunity to either continue without saving, continue after saving, or cancel the action. This notification will appear before generating new sentences, opening a saved session, or closing the software.

There is one other time when translations could be lost. If the program crashes without giving the user a chance to save their work, it will be lost. We have included an auto save functionality that, whenever a user changes the current sentence, saves the current session into an automatically generated recovery file. The user will then be able to open this file on their next load of the program should it crash.

The remainder of buttons and sliders at the bottom of the screen have to do with sentence generation. The "Generate" button triggers the program to generate and output sentences to the main window. The other objects allow the user to select various options regarding the generated sentences. First, the "Number" slider allows the user to specify the number of sentences that should be output. The "Complexity" slider determines the complexity of the sentences generated. At present, this simply determines the maximum length of generated sentences, but in future versions it will take into account other variables as well. The "Weighted" checkbox enables the user to choose whether or not sentences' vocabulary selection will be based on word frequencies. Finally, the radio buttons which switch between

"Semantic" and "Modelled" determine the type of generation to be used. We describe these two types of generation briefly in Section 2.3, and expand on them in Chapters 3 and 4. The processes that the various parts of the GUI run are shown in Figure 2.2.

In all, our hope for the GUI was to design an interface with which the user may begin working with the system immediately. We will continue to fine-tune it, and, as new features are added, likely group functionalities into other designs.

2.2 The Sentence Generation Algorithm

When the user clicks on the button that makes the system generate sentences, all they notice is a delay, followed by the sudden appearance of a sentence in the top box. However, there is a great deal of work happening behind the scenes during this delay. When the system triggers sentence generation, the main algorithm runs. Figure 2.3 gives an overview of the steps in this algorithm, which will be described throughout the remainer of the text.

The system allows the user to select between two types of sentence generation. There are two differences between the models: when sentence models are generated and the method of sentence model generation. The latter will be covered in subsequent chapters; here, we examine when sentence models are generated.

Both methods of generation follow a similar process after the sentence model has been chosen. The sentence model will be of a form similar to that shown in Figure 2.4. The model is a list of "words"; words are a list of feature structures. These feature structures contain the data required by the algorithm to form the final, conjugated words returned as part of the generated sentences. Features can be several things, depending on the language. The full list of features used in generating Greek sentences is given in Figure 2.5. One who knows Greek can see that not all of these can coexist in every Greek word; indeed, not every feature needs to have a value filled for the algorithm to run. In the case that a feature is left blank, the algorithm chooses a legal random value to place in the feature. Feature structures of one word can also be made to agree with other words' feature structures, so as to create grammatically correct sentences.

When the sentence model is filled, then the system will replace each "word" of the model with an actual Greek word. Much more will be said about this process and that of feature structure construction in Chapter 5. The algorithm replaces each word in the model, and when finished, returns the generated sentence. The entire process is repeated for each sentence that must be generated, based on how many the user requested. When finished, a list of the generated sentences is returned, and the user may then view them on the GUI.

2.3 Modelled vs. Semantic Generation

The next two chapters will provide in depth descriptions of both the modeled and semantic generation methods. Therefore, in this section we will simply provide a brief definition of both methods, as well as the benefits and drawbacks of using each.

The first method is modeled sentence generation. This method chooses sentence models from a set of premade models included with the system. The models are taken directly from

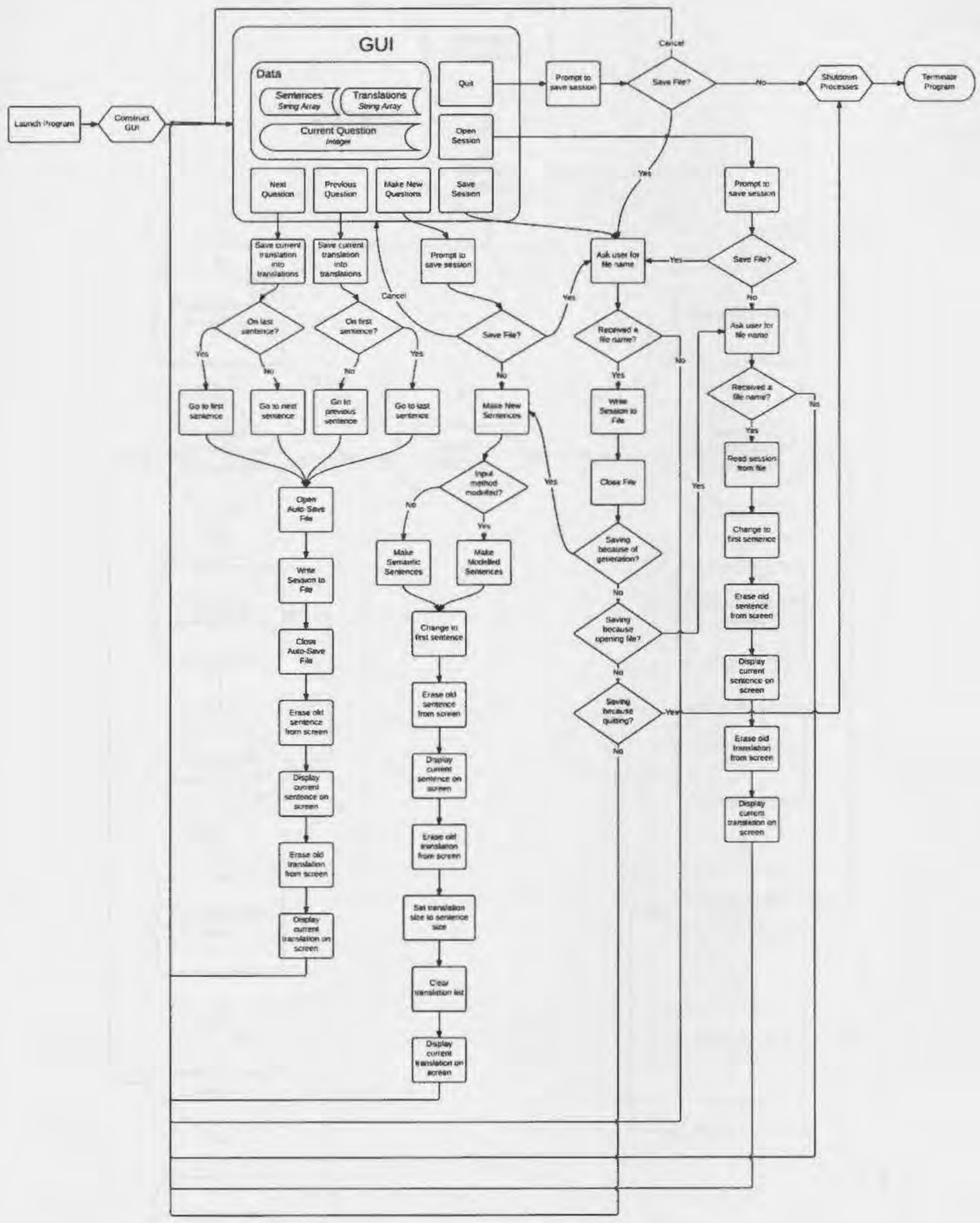

Figure 2.2: GUI Processes

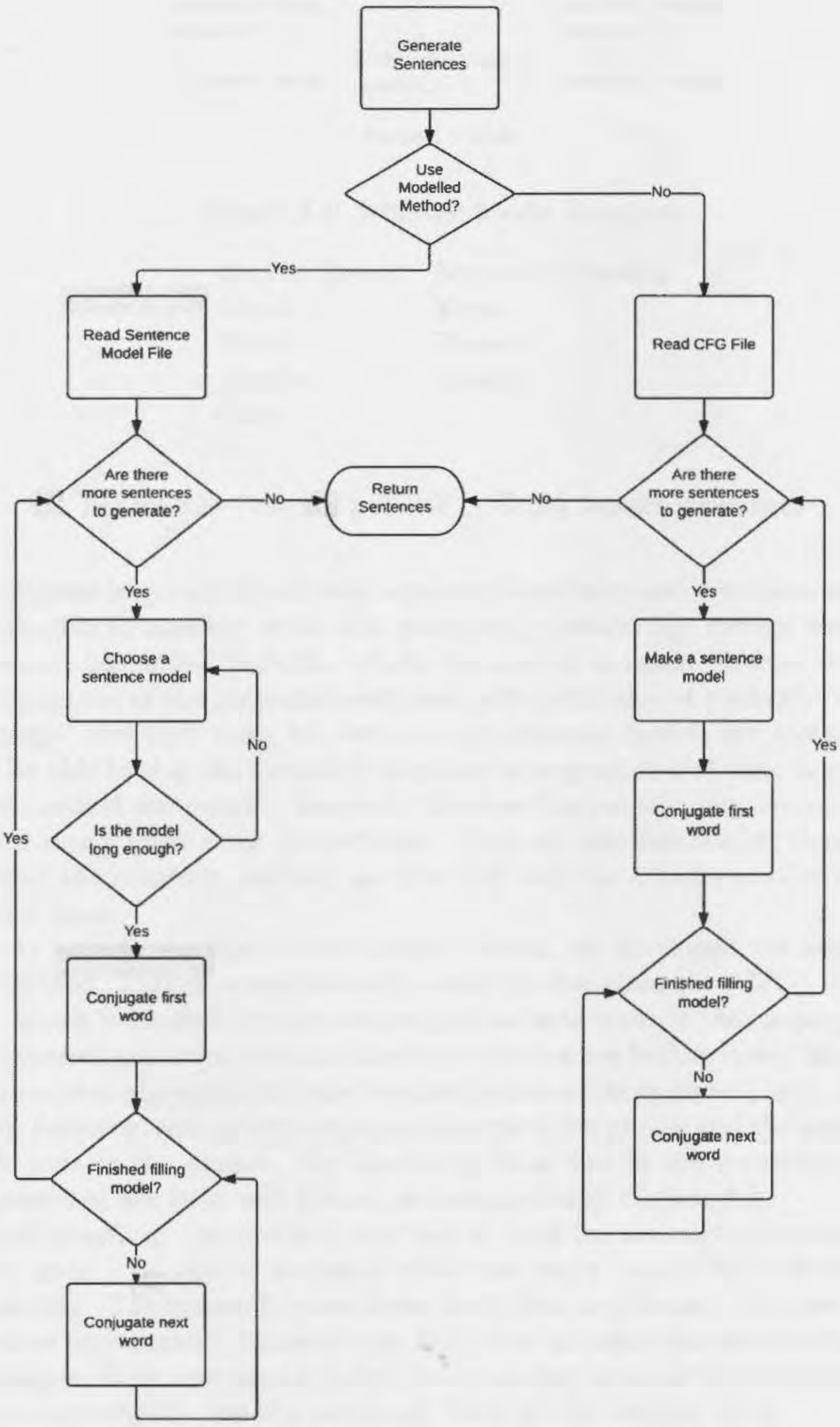

Figure 2.3: Steps of the Sentence Generation Algorithm

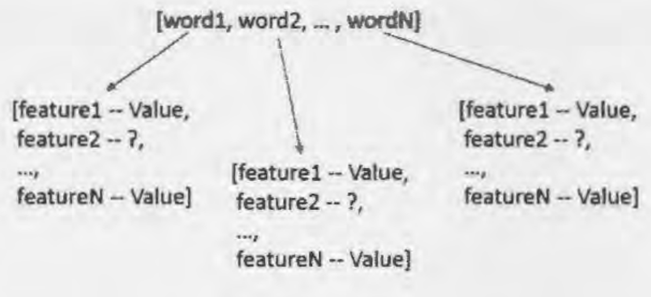
[word1, word2, ... , wordN]
[feature1 -- Value, feature2 -- ?, ..., featureN -- Value]
[feature1 -- Value, feature2 -- ?, ..., featureN -- Value]
[feature1 -- Value, feature2 -- ?, ..., featureN -- Value]

Figure 2.4: Sentence Model Structure

Part of Speech	Semantic Meaning
Mood	Tense
Voice	Person
Number	Gender
Case	

Figure 2.5: Features possible in Greek feature structures

text in the original language; the system analyzes these texts and generates sentence models that are as flexible as possible, while still generating syntactically correct sentences.

This method has a few benefits. First, because it is taken directly from the source language, the syntax of the generated sentences will match almost perfectly with that of the target language. Not only that, but because the sentence models are already formed, the system will be able to skip the formation step, saving a great deal of time in generation. The flaws of this method are notable, however. Because the models only represent syntax, the sentences will rarely make sense semantically. They are also less flexible than the sentences generated with the semantic method, as there will only be a finite number of models with which to draw from.

In order to remedy the flaws of the former method, we developed the semantic sentence generation method. This process starts with a context-free grammar (CFG), distributed with the system, which is created from annotated phrase structures of the language. The phrase structures represent the ways in which existing sentences are broken down into smaller parts, allowing a recursive algorithm to build custom sentences from these parts. Each phrase is annotated by features required by one or more parts of the phrase and the semantic meaning of the head node in the phrase. By combining these two in the recursive algorithm, the sentences generated are both well-formed and semantically meaningful.

The major benefit of this method over that of modeled sentence generation is that sentences make sense. Generated sentences also have more variety than those formed from pre-made models. The tradeoff comes from both time and detail. For one, the semantic algorithm takes considerably longer to run than does an algorithm for selecting a sentence. To a lesser degree, these sentences are also one more step removed from the source language, and therefore look slightly less like sentences found in the training data.

The two algorithms are can generate sentences in the target language, but the user must choose whether he or she wants to trade quality for time. The goal of the next chapters is to examine the methods we used to develop both the modeled and semantic algorithms.

Chapter 3

Modeled Sentence Generation

Our system contains two methods of sentence model generation. Of these, the first to be implemented was *modeled sentence generation.* This method, a much simpler process than its counterpart, semantic sentence generation, takes place before system distribution. Therefore, models are already made when tbe user runs the system, noticably improving run time.

Modeled sentence generation, though much faster than semantic sentence generation, suffers the drawback of creating sentences that make little to no sense semantically to the translator. Therefore, the user must determine whether quality sentences or speedy generation is of more importance.

The goal of this chapter is to outline the process of the modeled sentence generation algorithm and to describe the organization of the database in which the sentence models are held. Figure 3.1 shows how the system generates sentences using this method; this chapter is concerned with how the sentence model database is created.

3.1 Construction of the Sentence Models

The actual construction of the sentence models is fairly simple. A series of XML files detailing the sentences of our corpora, the bodies of text from which we draw, is used to find the features for each word in the corpus. These features are then analyzed by the system and replaced by a feature structure in the form of a string. Each feature structure is then considered a word, and the words are linked together as a list. These lists, now representing sentence models, are output to a 'Model" file, which contains the complete sentence model database.

The beauty of this algorithm comes from its simplicity; the pain comes from the organization of the corpora data. In order for this model to be used, the corpora must be hand (or automatically, which was not attempted in this project) labeled with the full range of features listed in the previous chapter. Therefore, it is important to understand the layout of the data files.

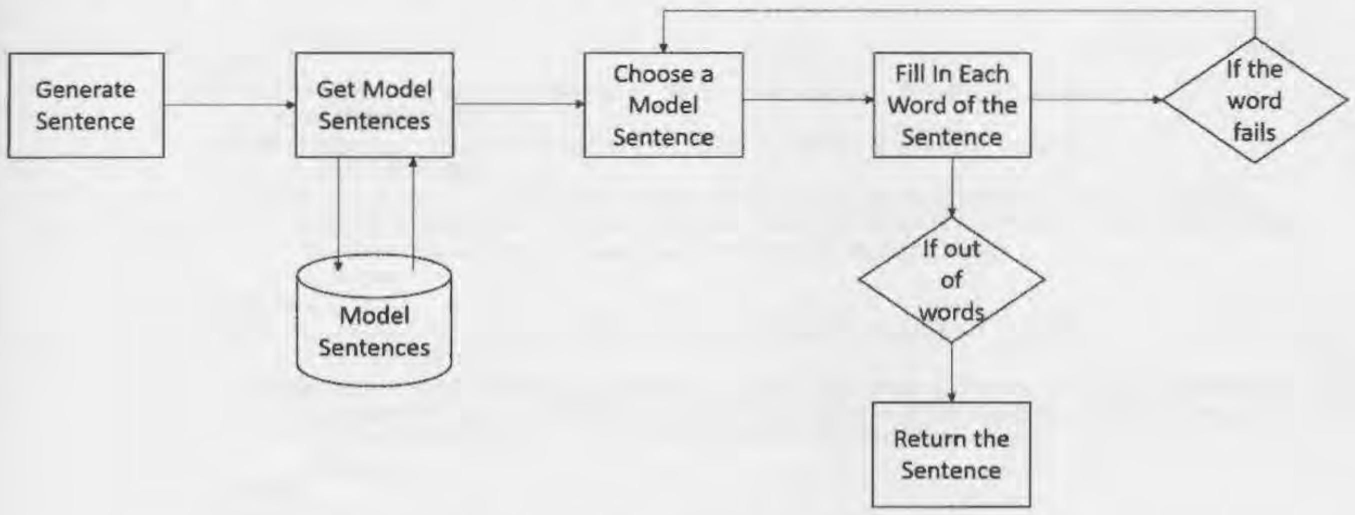

Figure 3.1: modeled sentence generation process

3.2 Database Design

In our algorithm, the features in the sentence models come from a series of XML files. Because we are implementing our project with Koine Greek, we used the books of the New Testament as our corpora (using the SBL version of the text). We knew that each sentence would need to be labeled with phrase structures, and each word would need to be labeled with both its part of speech and its features. We decided to use the label schema designed by the MorphGNT team. This group has painstakingly created fully labeled sentence diagrams of the Greek New Testament, complete with all the labels we would need. We had to enter semantic labels manually, as will be seen in the next chapter, but for modeled generation, this data set was exactly what we were seeking. Because it was implemented as XML files, split between the books of the New Testament, we decided to use this format for our work. Here is a sentence represented in XML format:

```
<Sentence ID = ''Mrk1:1:1-1:1:5'>
<Trees>
<Tree>
<Node Cat='S' Head='0' nodeId='410010010010053'>
  <Node Cat='CL' Start='0' End='4' Rule='P2CL' Head='0' ClType='Verbless' nodeId
      ='410010010010052'>
    <Node Cat='P' Start='0' End='4' Rule='np2P' Head='0' nodeId='410010010010051'>
      <Node Cat='np' Start='0' End='4' Rule='NPofNP' Head='0' nodeId='410010010010050'>
        <Node Cat='np' Start='0' End='0' Rule='N2NP' Head='0' nodeId='410010010010011'>
          <Node Cat='noun' Start='0' End='0' UnicodeLemma='ἀρχή' Unicode='Ἀρχὴ'
              Number='Singular' Type='Common' morphId='41001001001' Case='Nominative' Gender
              ='Feminine' nodeId='410010010010010'>Ἀρχὴ
              </Node>
        </Node>
        <Node Cat='np' Start='1' End='4' Rule='DetNP' Head='1' HasDet='True' nodeId
            ='410010010020040'>
          <Node Cat='det' Start='1' End='1' Case='Genitive' UnicodeLemma='ὁ' Unicode='τοῦ'
              Gender='Neuter' Number='Singular' morphId='41001001002' nodeId
              ='410010010020010'>τοῦ
              </Node>
          <Node Cat='np' Start='2' End='4' Rule='NPofNP' Head='0' nodeId='410010010030030'>
            <Node Cat='np' Start='2' End='2' Rule='N2NP' Head='0' nodeId='410010010030011'>
              <Node Cat='noun' Start='2' End='2' UnicodeLemma='εὐαγγέλιον' Unicode='εὐαγγελίου'
                  Number='Singular' Type='Common' morphId='41001001003' Case='Genitive'
                  Gender='Neuter' nodeId='410010010030010'>εὐαγγελίου
                  </Node>
            </Node>
            <Node Cat='np' Start='3' End='4' Rule='NP-Appos' Head='0' nodeId
                ='410010010040020'>
              <Node Cat='np' Start='3' End='3' Rule='N2NP' Head='0' nodeId
                  ='410010010040011'>
                <Node Cat='noun' Start='3' End='3' UnicodeLemma='Ἰησοῦς' Unicode='Ἰησοῦ'
                    Number='Singular' Type='Proper' morphId='41001001004' Case='Genitive'
                    Gender='Masculine' nodeId='410010010040010'>Ἰησοῦ
                    </Node>
              </Node>
              <Node Cat='np' Start='4' End='4' Rule='N2NP' Head='0' nodeId
                  ='410010010050011'>
                <Node Cat='noun' Start='4' End='4' UnicodeLemma='Χριστός' Unicode='χριστοῦ.'
                    Number='Singular' Type='Proper' morphId='41001001005' Case='Genitive'
                    Gender='Masculine' nodeId='410010010050010'>χριστοῦ
                    </Node>
              </Node>
            </Node>
          </Node>
        </Node>
      </Node>
    </Node>
  </Node>
</Node>
</Tree>
</Trees>
</Sentence>
```

In this format, sentences are labeled with an ID, the location in the New Testament from where it originated. The first three characters represent the book. The next number is the chapter, then following the first colon is the verse, and finally another colon and the word number. Then there is a hyphen, and the same series of information, without the book code. The first series tells where the sentence starts, and the second tells where it ends. So, in this case, the sentence begins with the first word of Mark 1:1, and ends with the fifth word of that verse.

The rest of the sentence is composed of a tree of phrase structures. Most of the labels in this tree are unimportant to modeled sentence generation, and will not be described until the next chapter. However, the algorithm does use the bottom leaf nodes, which represent words of the sentence, in generating models.

The model generator first goes through each leaf node, forming it into a feature structure. Each leaf node has a 'Cat", or category, attribute that is labeled with a part of speech. In generating a word in the model, this will be the first part displayed, before the feature structure. Then the system begins generating the feature structure, by searching for each of the features listed in the previous chapter, as attributes in the node. If any are present, these are appended to the feature structure. When each has been searched for, the part of speech and feature structure are combined, forming a 'word".

The final step is for the algorithm to put the words in the correct order. Because the data sets must organize the sentences so that words can correctly be placed in tree format, sometimes words, most notably postpositive conjunctions, are not displayed in their original order. Therefore, each leaf node is provided with a 'NodeID" attribute, which determines its position in the sentence. The first two digits represent the book, followed by another three representing the chapter, then three representing the verse, and finally, three representing the word index. Because these are inherently sortable, the system must simply take the set of words it has generated and sort them by the node ID. When this is done, the words

```
['noun_Case=1,Number=1,Gender=2', 'det_Case=2,Number=1,Gender=3',
    'noun_Case=2,Number=1,Gender=3', 'noun_Case=2,Number=1,Gender
    =1', 'noun_Case=2,Number=1,Gender=1']
```

Figure 3.2: Sentence model of the first sentence in Mark from the Models database

are then output as a sentence model to the main model database, and the next sentence is analyzed. Figure 3.2 shows an example line from the model database, representing a complete sentence. When read by the system, words can be separated by the quote-comma-space-quote delimiter, the part of speech found by looking to the left of the underscore, and the features by looking to the right of the underscore. Features are separated by commas, with their type given on the left of an equal sign, and their value given on the right.

Chapter 4

Semantic Sentence Generation

Modeled sentence generation was characterized by its pre-processed sentence models. *Semantic sentence generation*, on the other hand, will create sentence models as the algorithm runs. That is not the only difference between the models. Where the former generated sentences that made little sense semantically, the latter will create sentences that make more sense, as it takes into account semantic meanings while generating.

The drawback to these 'semantically correct" sentences is that it takes much longer for the system to generate them. As will be shown, model creation with the semantic algorithm requires a recursive examination of a CFG, and sometimes returns failed models and must begin again. Therefore, the user must choose whether they desire sentences created faster or sentences of better form. In a similar layout as the last chapter, this chapter aims to describe the semantic sentence generation process. It begins by giving a detailed description of how the algorithm runs, an overview of which is shown in Figure 4.1. Then it describes the databases used by the system in storing information needed to run the algorithm.

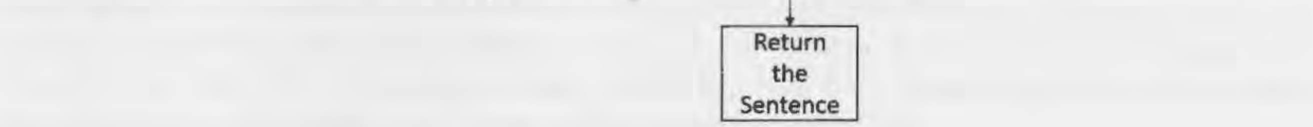

Figure 4.1: Semantic sentence generation

4.1 Construction of the Sentence Models

When the user chooses to generate a sentence using semantic sentence generation, the system first creates a navigable CFG from a premade grammar file, which we cover in the next section. For now, it will suffice to know that the grammar file is made of production rules, representing phrase structures. Each production rule has a head, representing the overall structure, and one or more children, representing the substructures that make up the entire phrase structure. Structures can be recursive, that is, they can include themselves as their own children. Each head and child is labeled with zero or more semantic domains and zero or more features.

The grammar file is then transformed into a CFG by taking each rule and adding it to a *production* dictionary, containing all of the production rules of the grammar. The keys of the dictionary are the names of the heads of the rules in the grammar along with their semantic domain, and the values for each head are a list of one or more child representations. This allows for one head rule to contain multiple sets of children. Child representations are tuples containing each child's head, the feature structure of the rule (because all nodes in a grammar rule must have the same feature structures), and the head of the rule, which determines which child passes features through the dictionary when it is the head of a phrase structure.

When this process is finished, the system may then traverse the grammar and recursively combine rules to create sentence models. For each sentence that is to be generated, the system calls a recursive algorithm to create a sentence model. The algorithm uses convergent probabilities, meaning that productions that have already appeared in a derivation of a branch have a smaller chance of being selected. This will lower the likelihood of recursive phrase structures occurring too often. The algorithm also uses a maximum depth variable that prevents it from recursing too deeply. This, in effect, cancels the algorithm if it will take too long or result in an infinite loop.

To create the sentence, the algorithm first chooses a starting production rule. Once chosen, it recursively calls itself on each child in the rule. Then, the algorithm chooses a production rule that has the child as a head. This is done repeatedly until the child is a terminal node, or one which is not the head of a node. In this grammar, the only nodes which are not the head of a node are *words* that can be added to the sentence model. The algorithm pays attention to domain labels and feature structures as it works its way down the tree. Both of these are passed up and down through *head* children, and the algorithm is only able to add production rules where the domains and features unify. If they do not, the system backtracks and tries a different rule. As the system backtracks upon finding these words, they are added to phrase models. At the top call, the main one called by the system, the phrase model returned will be the sentence model asked for.

At this point, the system must ensure that the model meets the complexity demands asked for by the user. If it is of sufficient length, the system can move on. If not, it must retry the recursive sentence model maker until it finds one of sufficient length.

Words of the system model are of a similar form as those produced in the modeled sentence generation method. They consist of a part of speech and feature structure. However, because the grammar relies partially on semantic labels to create production rules, the words also contain these labels. Therefore, when the system begins to conjugate the words, it uses this label alongside the feature structure and part of speech. This conjugation step is covered in the next chapter. After the words have been conjugated, the system presents the sentence to the user.

4.2 Database Design

The semantic sentence generation algorithm uses one database, the grammar file, when generating the models. We will now describe the creation and usage of this file.

Before distribution, the semantic sentence generation algorithm constucts modified versions of the sentence XML files discussed in Chapter 3. Before, we discussed only the child nodes of these sentences. Now, however, we will examine each of the nodes and their attributes in full. We again look at the first sentence in Mark, expressed in XML here.

```
<Sentence ID=‘Mrk1:1:1−1:1:5'>
<Trees>
<Tree>
<Node Cat=‘S' Head=‘0' Rule=‘S'>
  <Node Cat=‘CL' Head=‘0' Rule=‘P2CL'>
    <Node Cat=‘P' Head=‘0' Rule=‘np2P'>
      <Node Cat=‘np' Head=‘0' Rule=‘NPofNP'>
        <Node Cat=‘np' Head=‘0' Rule=‘N2NP'>
          <Node Case=‘Nominative' Cat=‘noun' Domain=‘Time' Gender=‘Feminine' Number=‘
              Singular' Rule=‘noun'>Ἀρχὴ
              </Node>
        </Node>
        <Node Cat=‘np' Head=‘1' Rule=‘DetNP'>
          <Node Case=‘Genitive' Cat=‘det' Domain=‘Article' Gender=‘Neuter' Number=‘Singular'
               Rule=‘det'>τοῦ
              </Node>
          <Node Cat=‘np' Head=‘0' Rule=‘NPofNP'>
            <Node Cat=‘np' Head=‘0' Rule=‘N2NP'>
              <Node Case=‘Genitive' Cat=‘noun' Domain=‘Information' Gender=‘Neuter' Number=‘
                  Singular' Rule=‘noun'>εὐαγγελίου
                  </Node>
            </Node>
            <Node Cat=‘np' Head=‘0' Rule=‘NP−Appos'>
              <Node Cat=‘np' Head=‘0' Rule=‘N2NP'>
                <Node Case=‘Genitive' Cat=‘noun' Domain=‘Name' Gender=‘Masculine' Number=‘
                    Singular' Rule=‘noun'>Ἰησοῦ
                    </Node>
              </Node>
              <Node Cat=‘np' Head=‘0' Rule=‘N2NP'>
                <Node Case=‘Genitive' Cat=‘noun' Domain=‘JobTitle' Gender=‘Masculine' Number
                    =‘Singular' Rule=‘noun'>χριστοῦ
                    </Node>
              </Node>
            </Node>
          </Node>
        </Node>
      </Node>
    </Node>
  </Node>
</Node>
</Tree>
</Trees>
</Sentence>
```

In these models, we show far fewer attributes. As opposed to the files used in modeled sentence generation, however, each of these attributes plays an important role in constructing the grammar. The algorithm only ignores the ID, which is simply a reference to the verse where it is found.

While the modeled sentence generation algorithm created separate models for each sentence in the corpus, the semantic sentence generation uses data from the overall syntax structure of each sentence to construct sentences. To do this, it constructs the grammar from all of the sentences.

The model examines each node in each tree and creates production rules from them. It first takes the *Cat*, or category, attribute, and combines it with the *Rule* attribute. It then looks at the *Cat* and *Rule* attributes of each child of the node and adds them to the rule's children. If a child is a terminal node, it is also given the domain label and feature structure of the node. This forms the basic structure of grammar, but with the rules still in the original tree form.

Because it is still in tree form, the features and domains of terminal nodes may easily be passed up the tree. Therefore, the next step works from the bottom up in each tree, combining all feature structures and domains agreed upon by sibling nodes. So, for example, a rule with two children, one being masculine, singular, and nominative and the other being feminine, singular, and nominative, will then be labeled with the domain and a generic feature structure, and the head child will be labeled. The feature structure in this case represents that the children must agree in number and case, but not necessarily gender. The head child passes domains and feature structures up and down the tree; two heads must be able to unify for the rules to coexist in the sentence. The head child is labeled with a *head* keyword attached to the appropriate child in the rule.

When all production rules are labeled correctly, they are ready to form the grammar file. The system begins by forming a set. It traverses through each rule, adding it to the set. Because duplicates are not allowed in sets, this ensures that the grammar will not be redundant. Then, the rules are output into a grammar file to be used by the generation algorithm. Below is a sample grammar taken only from the first sentence in Mark that shows the final format of our database.

```
S[Domain='Time'] —> P2CL-Head[Domain='Time']
P2CL[Domain='Time'] —> np2P-Head[Domain='Time']
np2P[Domain='Time'] —> NPofNP-Head[Domain='Time']
NPofNP[Domain='Time'] —> N2NP-Head[Domain='Time'], DetNP[Domain='
    Information']
N2NP[Domain='Time',Case='?c',Gender='?g',Number='?n'] —> noun-
    Head[Domain='Time',Case='?c',Gender='?g',Number='?n']
noun[Domain='Time',Case='Nominative',Gender='Feminine',Number='
    Singular'] —> {noun_Domain='Time',Case='Nominative',Gender='
    Feminine',Number='Singular'}
DetNP[Domain='Information',Case='?c',Gender='?g',Number='?n'] —>
    det[Domain='Article',Case='?c',Gender='?g',Number='?n'], NPofNP
    -Head[Domain='Information',Case='?c',Gender='?g',Number='?n']
det[Domain='Article',Case='Genitive',Gender='Neuter',Number='
    Singular'] —> {det_Domain='Article',Case='Genitive',Gender='
    Neuter',Number='Singular'}
NPofNP[Domain='Information'] —> N2NP-Head[Domain='Information'],
    NP–Appos [Domain='Name']
N2NP[Domain='Information',Case='?c',Gender='?g',Number='?n'] —>
    noun–Head[Domain='Information',Case='?c',Gender='?g',Number='?n
    ']
noun[Domain='Information',Case='Genitive',Gender='Neuter',Number='
    Singular'] —> {noun_Domain='Information',Case='Genitive',
    Gender='Neuter',Number='Singular'}
NP–Appos[Domain='Name'] —> N2NP–Head[Domain='Name'], N2NP[Domain
    ='JobTitle']
N2NP[Domain='Name',Case='?c',Gender='?g',Number='?n'] —> noun–
    Head[Domain='Name',Case='?c',Gender='?g',Number='?n']
N2NP[Domain='JobTitle',Case='?c',Gender='?g',Number='?n'] —> noun
    –Head[Domain='JobTitle',Case='?c',Gender='?g',Number='?n']
noun[Domain='Name',Case='Genitive',Gender='Masculine',Number='
    Singular'] —> {noun_Domain='Name',Case='Genitive',Gender='
    Masculine',Number='Singular'}
noun[Domain='JobTitle',Case='Genitive',Gender='Masculine',Number='
    Singular'] —> {noun_Domain='JobTitle',Case='Genitive',Gender='
    Masculine',Number='Singular'}
```

Chapter 5

Filling the Sentence Models

After the sentence model has been generated, either by modeled sentence generation or semantic sentence generation, it must be filled in with grammatically correct words in the target language. The process to create these words is universal, though the modules will need to be changed in order to fit the target language, in our case Greek.

The process of choosing and conjugating words can be split into two parts. First, a list must be created of potential words. This is done in the same way by each part of speech. Second, a word is randomly chosen from the list and put into the grammatically correct form specified by the system. Figure 5.1 shows this flow of events. This process is done for every word in the sentence model. If a word fails to conjugate, the sentence model fails and the system backs up to choose a new one.

This chapter is organized into three sections. In the first, the process of making a list of words is discussed. Then, we explain the various databases which hold information about the words. Finally, we examine the second half of the word creation algorithm, in which the words themselves are created.

5.1 Word Choice Process

The first half of the word conjugation algorithm creates a list of potential words. To do so, the system takes the part of speech of the word and opens the corresponding lexical database, essentially a dictionary. There are eleven lexicons, each corresponding to a part of speech.

For modeled sentence generation, the process is simple: every word in the database is added to the list. For semantic sentence generation, however, the algorithm makes one adjustment. Because this method takes into account word meaning, it must form another list, comprised of all words of the specified domain, and take the intersection of the two lists.

From the list that is created, a random word is chosen and a part of speech dependent function is run to conjugate the word, which will be discussed in Section 5.3. If the function fails to conjugate a word, that word is removed from the list. A random word continues to be chosen until either the list is empty or a correct word is supplied. In the case of the former, the sentence generator must backtrack and create a new sentence model. If the word succeeds, however, the algorithm saves the word and moves on to the next position. When

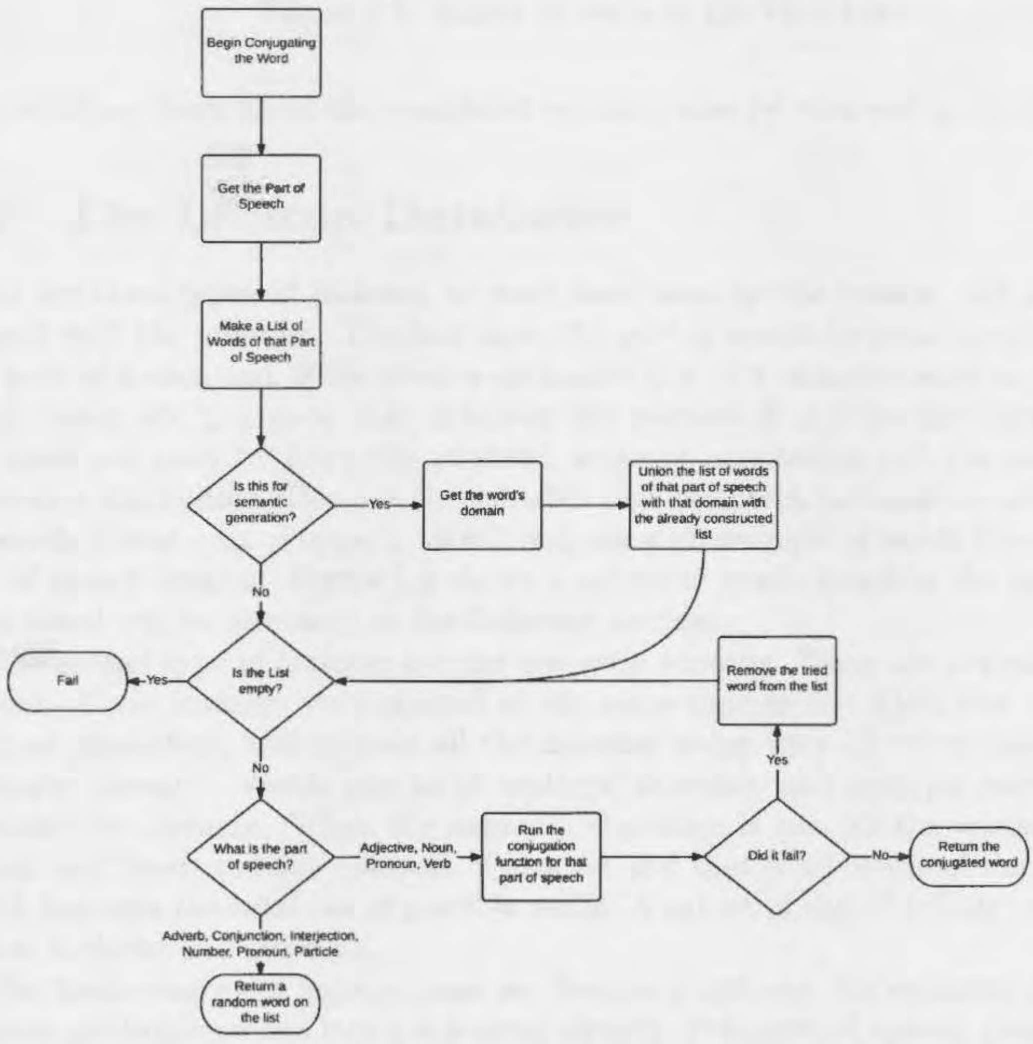
Begin Conjugating the Word
Get the Part of Speech
Make a List of Words of that Part of Speech
Is this for semantic generation?
Yes
Get the word's domain
Union the list of words of that part of speech with that domain with the already constructed list
No
Is the List empty?
Yes
Fail
No
What is the part of speech?
Adjective, Noun, Pronoun, Verb
Run the conjugation function for that part of speech
Did it fail?
Yes
Remove the tried word from the list
No
Return the conjugated word
Adverb, Conjunction, Interjection, Number, Pronoun, Particle
Return a random word on the list

Figure 5.1: Choosing and conjugating a word in the sentence model

cv1c	διαμένω
cv3a	ὑπολιμπάνω
cv1b	καταλείπω
cv1b	ἐγκαταλείπω
cv1b	παράγω
cv1b	παρέρχομαι
cv1b	ἀπέρχομαι
cv1b	ἐξέρχομαι

Figure 5.2: Subset of verbs in the Verb Lexicon

all words have been filled, the completed sentence may be returned to the GUI.

5.2 The Lexicon Databases

There are three types of lexicons, or word lists, used by the system. All are premade and shipped with the program. The first ones, the *part of speech lexicons*, consist of all words of that part of speech and, if the word is declinable (i.e. if it changes based on number, gender, mood, tense, etc.), a code that indicates the method it will use for conjugation. These databases are used by both the modeled sentence generation and the semantic sentence generation algorithms. Because indeclinable part of speech lexicons are simply a listing of the words of that part of speech, we will only show an example of words found in a declinable part of speech lexicon. Figure 5.2 shows a subset of words found in the verb lexicon. The codes listed will be discussed in the following section.

The second type of lexicons are the *semantic lexicons*. These are organized by semantic domain. These lexicons were created at the same time as the XML files used in semantic sentence generation, and contain all the domains along with all words that can be of that particular domain. Words can be of multiple domains, and multiple parts of speech can be within one domain. When the semantic algorithm is run, all the words of the specified domain are listed, and the intersect of this list and that of all words of the specified part of speech becomes the total list of possible verbs. A subset of the "JobTitle" semantic domain lexicon is shown in Figure 5.3.

The final category of lexicons used are *frequency lexicons*. For semantic domains, the frequencies are implemented into the lexicon already. For parts of speech, there are a separate set of lexicons containing the frequency of each word of each part of speech. These are only used if the user chooses to generate weighted sentences. Instead of choosing a completely random word when choosing from the list, if the user wants weighted sentences, more common words will be given a bias over less common words. This feature is still undergoing experimentation and is not working as well as we had hoped. These lexicons are structured in the same way as the semantic lexicons, except that they appear with their part of speech instead of their domain.

διάβολος:10
πρεσβύτερος:24
ἐγκάθετος:1
μίσθιος:2
χήρα:12
ἀλλότριος:2
φιλάργυρος:1
κακοῦργος:3
μισθωτός:1
νομικός:6
περίοικος:1
ἀλλογενής:1
μέτοχος:1
ἀλλοτρίων:1

Figure 5.3: Subset of words found in the "JobTitle" domain, along with their frequencies

5.3 Part of Speech Conjugation

We distinguish eleven parts of speech in the Greek language for the purposes of our program. Of these, six are indeclinable (adverbs, conjunctions, interjections, numbers, particles, and prepositions). This section explains how we implemented each of the five declinable part of speech conjugation algorithms. When searching for an indeclinable part of speech, a word is simply chosen at random, provided the list contains a word that can be chosen.

All of the conjugation functions require three inputs. First is the word to be conjugated. Second is the code, which comes from the speech lexicon. Finally, the function requires a complete feature set for the word it is conjugating. Because the generation algorithms allow words to contain any non-contradicting features (i.e. two or more features that contradict and so do not allow a word to be formed), the conjugation functions were placed inside of helper functions, each of which takes any number of features, and randomizes the unspecified features to ones that legally work with the ones given. It also ensures that it has not been given contradicting features. Therefore, the algorithm can assume that it will not fail due to, for instance, a plural vocative or a first person imperative.

The codes used in the speech lexicons, some of which are shown in Figure 5.2, are derived from the morphological codes used by Mounce[23]. When constructing the functions, we analyzed the behavior of each code and were able to systematically derive conjugation steps for each word based on their codes.

5.3.1 Verbs

Verbs were the most complex type of word we had to conjugate. The verb generator was divided into over one hundred sections, depending on the mood, tense, and voice of the verb. Then, these sections were subdivided even further, based on the conjugation code. Even at this level, nuances in certain types of words required branching statements to ensure that every type of verb was handled correctly.

5.3.2 Nouns

Nouns proved to be easier to tackle than verbs, but not by much. They were divided only on their codes, because their codes had already been divided on declension. These, too, had many nuances that we had to work around, though there was less branching that had to be done.

5.3.3 Adjectives

Most adjectives were more regular than nouns when conjugating. We found that there was less branching than most other parts of speech, and when they did branch it was usually only because of slight differences.

5.3.4 Pronouns

Pronouns were by far the most irregular of the parts of speech. Because there are far fewer of these than verbs, they were less complex to program, though we found that for nearly all of them, we had to handle them on a case by case basis.

5.3.5 Determiners

Determiners were by far the easiest class of words to decline, as we only had to handle two cases: the article ὁ and the relative pronoun ὅς . Therefore, we simply structured this as a single if-elseif statement and handled the conjugation of each word in turn.

Chapter 6

The Implemented System

We have now looked at each part of the sentence generation algorithm in detail, and have explored the GUI that enables the user to generate sentences and translate them. Now, we examine the implementation of the system we developed. We do this in two phases. First, we explain why we chose the language and corpora included with the first version of the system. Then we explore the development process, starting at the research stage and ending with the finished product.

6.1 Corpora Choice

The first version of the system comes ready to work with a single language, Koine Greek, based on a corpora consisting of the *Society of Biblical Languages* (SBL) version of the text of the New Testament[16]. We chose the target language for two reasons. First, because we have seen and participated in the study of New Testament Greek firsthand, we realized that students could benefit from seeing new, well-formed sentences which could be used for practice. The second reason comes from a more pragmatic perspective. We knew of Mounce's work, *The Morphology of Biblical Greek*[23], and realized that it could make the process of creating conjugation functions much simpler than a language without a similar work to help guide the development process.

Once we decided on the language, the choice of corpora was obvious. Because we were studying the language of the New Testament, for the purpose of helping students of the New Testament, it only made sense to use this body of literature as the data for the system. When we discovered the work of the MorphGNT group[21], we realized that we had made a fantastic choice of corpora. By using their work as a basis, we set about organizing the data and getting it into the format needed for our project.

One note must be made on the usage of the corpora. Because of time constraints, we were not able to label all the books of the New Testament semantically. At this point, only Matthew through John are labeled completely, though most of the rest of the text is labeled partially. Because we can only use completely labeled texts in the semantic sentence generation method, it only relies on the first four books. However, these constitute a majority of the New Testament, because nearly all of the remaining books are only a fraction of the size of the first four. This does not affect the modeled sentence generation method, however, because all of the data is correctly labeled for attributes needed in the method.

6.2 Development Process

Development started, like any project, with a large amount of research. We first explored many important texts on both NLP and NLG. Though we found that our work would be a new contribution to the field, we analyzed related projects, trying to glean from them the best process to follow when developing and implementing our system. During the research phase, we were also fortunate to meet with some of the top names in natural language generation, and were able to get their input on our work.

After research, we began designing the architecture of our software. Knowing that our project would eventually be extended to multiple languages and corpora, we designed the system with a module-based architecture. The main sentence generation algorithm would run regardless of the language, as would the two model generation methods. What would be exchanged, based on the language and corpora choice, would be the lexicons, sentence data, grammar, and conjugation rules and algorithms. Therefore, we designed the process to allow substitutions of one or more of these databases.

The next step was to label the data and create the databases. Fortunately, the only changes that needed to be made to the XML files were syntactic, in order to get them into the format necessary to run with our algorithm. We wrote a few simple scripts to get them into the needed format. Then, we wrote a function to match words in the XML files with the conjugation codes given in an electronic version of Mounce's work, creating our part of speech lexicons. We also created a method to count each word in the XML files and create our frequency databases. The last step was the most time consuming; we labeled each word of the New Testament with a semantic label from a custom tag set and merged them with the XML files. As mentioned above, we were only able to label the first four books of the New Testament in this format. Then, we used a similar counting function as before and created our semantic lexicons.

We then created the data to be included with the system. First, we created the function described in chapter three to construct the sentence model database, for use with the modeled sentence generation function. Then, we created the grammar, using the process described in chapter four, to be used in the semantic sentence generation method.

By far, the most time consuming part of the development process was the creation of the conjugation functions. While Mounce's work laid out the conjugation process of each type of word, we discovered that there were many small nuances that needed to be handled in order to ensure grammatically correct conjugation. We spent a great deal of time writing these functions, and when finished, were finally able to move on to the overall development of the system.

When writing the sentence generation algorithm, we first focused on writing it based on the modeled sentence generation method. We went through several rewrites of the code before settling on a format that would work for our purposes. With this finished, we were able to generate our first sentences. We then turned our attention to the semantic sentence generation method. Again, we went through several rewrites before getting it right, but because many of the problems had been addressed in developing the modeled method, this process was simpler.

The final step was to develop the GUI of the system. As we were usiug the Python language to write the source code, we explored several libraries for GUI development. We decided to use Tkinter, in part because of the availability of more tutorials and examples than other options. Once the GUI was developed, we spent the remainder of our time tweaking existing processes and making small improvements to the system.

As can be seen, the development of our system was a lengthy process. It involved countless hours of both research and coding, though the final product is a novel product and area of research in the overall field of NLG. In the next chapter, we will analyze the performance of our software compared to our hopes, as well as our plans for future work.

Chapter 7

Conclusion

When we finished developing the system, we had software that did what it was intended to do: generate sentences using one of two generation methods. This chapter will analyze the results of the generation compared with our hopes for the system. It will also discuss remaining work that must be done to optimize the current algorithms and future work we have planned for the system.

7.1 Results

While our system managed to create sentences successfully, the ones produced were of poorer quality than we had hoped. Most sentences do generate in a grammatically correct forms, but there are more errors than we had hoped. The semantic sentence generation method also does not perform optimally when choosing word meanings.

The test results gave us direction on where our attention should turn for optimizing and tweaking the system. The rest of the work that needs to be done can be split into two parts. The remaining work is both work that we discovered must be done when testing the system and work that we did not have time to finish. Future work is work that we knew from the beginning we would be not able implement in this first version, but that we plan to add in future versions.

7.2 Remaining Work

There were many issues that made themselves apparent in the development and testing processes. There were two developmental processes we were unable to finish. First, we were not able to finish labeling all the words in the corpora with semantic domains. Before opening the software to the public, this must be completed. Second, another issue with the semantic part of the project, we were unable to implement a hierarchical choosing of semantic domains. Ideally, when choosing semantic domains for words, any domain that is of the specified domain or of a subdomain of the specified domain should be allowed. For instance, if the domain is "Person", domains of "JobTitle", "Name", and "PersonReferral" should be allowed as well. Currently, the system only allows domains that are of the same domain, not subdomains. Fixing each of these issues should improve the performance of the semantic sentence generation method, since the presence of subdomains will provide the system with a richer choice of words.

Several other issues arose in testing the system. First, we must go through and optimize each algorithm used. In their current state, there are many redundant processes being run, and many processes that can be improved. This should greatly increase the speed of the program, which will be better for user interaction. The algorithm for choosing sentence complexity should also be improved. Instead of making complexity the equivalent of minimum sentence length, as it now does, it should be related to other factors, including variety of phrase structures, vocabulary, and grammar.

Another issue arose when testing the weighted generation feature. Instead of giving a slight difference in appearance rates between words of different frequencies, there was a major difference. This resulted in the modeled sentence generation method returning sentences with most words of the same part of speech returning the same root word. In semantic generation, the most common words were nearly always chosen, with those that are only slightly less common occurring very rarely.

Finally, the conjugation functions should be improved. As they are, they work better than expected for conjugating individual words, though they still make several fixable errors. Not only should the functions for conjugating individual words be improved, but also functions for working with groups of words should be added. There are currently no considerations made for surrounding context when conjugating words, meaning that capitalization, assimilation, and compounding are given no attention.

7.3 Future Work

Aside from the issues made apparent in development and testing, there were several issues that we knew would have to be put off until future versions. Foremost among these is the inclusion of additional languages. We want our software to work for students of a variety of foreign languages, not just Koine Greek. Therefore, we plan on adding additional language functionality. The issue with this two-fold. First, in order to add additional languages, we must have one or more scholars of that language to consult with during development and who can test the result. Second, we must create the same types of databases used for Greek in corpora of the target language. This is no trivial task, especially without the resources we were able to attain in the development of the Greek databases.

We not only want to expand languages, but also corpora. In order to produce better sentences, we want to include other literature from first century Greek. This will widen the variety of grammatical structures and vocabulary in the sentences generated, as well as gain more accurate frequency counts.

In future versions of the system we also want to incorporate multithreading into the generation methods. This will improve the speed of generation by allowing multiple sentences to be generated at the same time and by letting more than one sentence generation process happen simutaneously.

Finally, future versions of the system will be translated into a more efficient programming language. Python made for faster coding, but the nature of it as a scripting language means that it cannot run at the speed of a compiled language like Java or C. Before making the software publically available, this step will have to be done.

Appendix A

Getting Started Guide

Biblical Greek Practicer is a program designed to help students practice translating by providing them with automatically generated sentences in the style of New Testament Greek. This document will guide the user through the installation of the program, the layout of the user interface, and the basic operations that can be performed.

A.1 Installation

In order for the program to run, the user must have Python 2.7 installed. To get it, visit `https://www.python.org/downloads/`. Ensure that the most recent version of Python 2.7, not a version of Python 3, is installed. Once downloaded, run the installer and follow the instructions provided.

After installing Python 2.7, obtain the BiblicalGreekPracticer.zip file, either from `http://www.tobeupdatedwebsite.com/downloads/` or the program cd. Extract the contents to the desired program location. Run the program by double clicking the BiblicalGreekPracticer.py file in the top folder, or, if Python files are not set to open with the Python Launcher by default, by right clicking the file and select Open With>Python Launcher. This will run the program.

A.2 Layout of the Interface

Figure A.1 shows a screenshot of the program. The top text box is where generated sentences will be displayed. The bottom text box is where the user can input translations to other sentence. On the bottom of the screen is a control panel that allows the user to navigate between sentences, save and open sessions, and generate new sentences.

A.3 Creating New Sentences

To create new sentences, the user should press the "New" button on the bottom right of the interface. This will create sentences based on the default options. To change the default options, the user can adjust the settings to the left of the new button. The "Number" slider

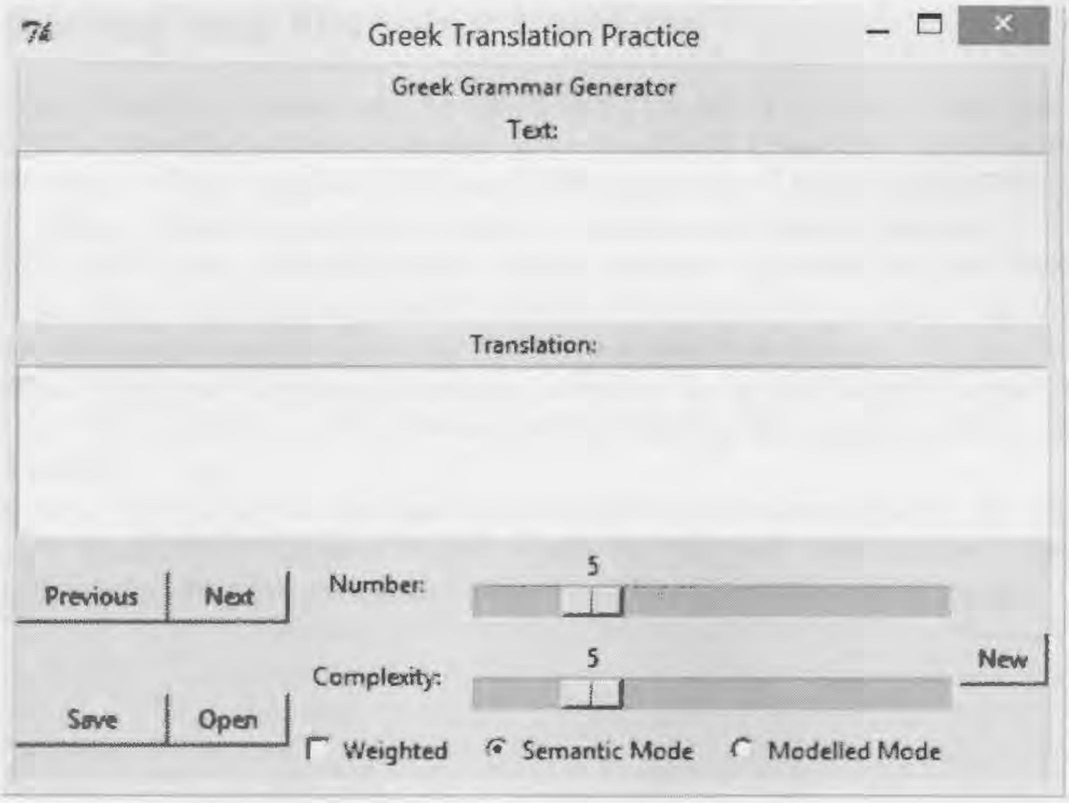

Figure A.1: Layout of the Interface

determines how many sentences to generate. The "Complexity" slider changes the minimum number of words in the generated sentences. The "Weighted" checkbox enables the generation of sentences that take vocabulary frequencies into consideration. Sentences created with this option will have common vocabulary words used in the generated sentences more often than less common vocabulary. The "Semantic" and "Modelled" radio buttons allow the user to switch between semantic sentence generation and modeled sentence generation. The former method will generate sentences from a grammar and a semantic domain lexicon. The latter creates sentences from sentence models generated based on real sentences in the corpora. When the sentences are generated, the first is displayed in the top box. The user can now move between sentences and translate them.

A.4 Navigating Between Sentences

At the bottom left side of the screen are two buttons, labeled "Next" and "Previous", which allow the user to navigate between generated sentences. If the last sentence is selected and "Next" is clicked, the program wraps around to the first sentence. Likewise, if the first sentence is selected and "Previous" is selected, the last sentence is displayed.

When the user changes to a new sentence, the system auto-saves the session. More will be said about auto-saving in the next section.

A.5 Saving and Opening Sessions

Biblical Greek Practicer sessions can be saved and opened. The session files will have the extension ".fll". In order to save a session, the user should press the "Save" button in the bottom left corner of the window. To open a file, press the "Open" button located beside the "Save" button. When a session is opened, it replaces the current session.

A session ends if the program exits, a saved session is opened, or new sentences are generated. In order to prevent unwanted deletion of sessions, the program will prompt the user to save the current session when one of these actions is performed. It is not required to save a session; if the user chooses to not save a session, he or she should choose the "Don't Save" option. To cancel the action chosen without ending the current session, choose the "Cancel" option.

In case of a system crash, the user may restore the previous session by opening the "autosave.fll" file in the "Autosave" folder where the program was installed. Sessions are automatically saved after the user clicks either the "Next" or "Previous" button.

Appendix B

Semantic Sentence Generation Pseudocode

This appendix contains the pseudocode for the recursive algorithm that produces sentence models using semantic sentence generation. We chose to include this function because it is the least straight-foward part of our work, and we hope that by including it future researchers will be able to avoid the struggles we encountered as we developed it. The remainder of this appendix contains the algorithm.

```
GenerateSemanticSentence( cfg, symbol, domain, featureStructure,
  countFactor, productionCount, depth )

// cfg -- A CFG object that contains a dictionary of production
   rules. The keys are tuples made of the symbols and domains of
   the head nodes, and the value of each key is a list of
   productions for that symbol. The elements of the list of
   productions are a tuple in the form (right-hand-side,
   production-feature-structure, child-head-number). This way,
   there can be more than one rule for each head symbol.

// symbol -- Symbol of the head node of the current production
   rule (should start as the grammar's start symbol)

// domain -- Domain of the head node of the current production
   rule. If null, a random one (that is possible for the symbol)
   is chosen

// featureStructure -- Feature sructure needed to unify with the
   symbol. Should start as null. Only passes down through the head
   child of the production.

// countFactor -- Controls how tight the convergence is. 0.0 <
```

```
   countFactor < 1.0. The higher the value, the more likely
   recursive phrase structures (clauses of the same type within
   each other) can occur. Used alongside productionCounts to
   determine which production rule to choose.

// productionCounts -- Dictionary that sets the value of new
   entries to 0. Contains the number of times productions have
   occured in the current branch. Used alongside countFactor to
   determine which production rule to choose. The keys are the
   head nodes in the production rule.

// depth -- Maximum depth to recurse. If this dips below 0, the
   algorithm fails.

if depth > 0 then

subsentence = []

// Select a domain if none is given

if domain == null

// Generic function to choose a legal domain for a given
   symbol

domain = ChooseRandomDomain(symbol)

end if

// Select a random production of the symbol that can unify
   with the feature structure that was passed down. If no
   production can unify, the algorithm fails

productions = cfg.productions[ ( symbol, domain ) ]
randomIndex = RandomInteger(0,productions.length-1)
randomProduction = productions[randomIndex]
productions.delete(randomIndex)

// Make the new feature structure (that will be passed down)
   by unifying the passed down feature structure with the
   first child of the production rule

newFeatureStructure = unify(featureStructure,
randomProduction[1])
```

```
// While the new feature structure was not able to be formed
   (unification was not possible), but there was a child in
   the production and there were still productons in the list,
   choose a new random production

while newFeatureStructure == null and
      randomProduction[1] != null and
      len(productions) > 0 do

randomIndex = RandomInteger(0,productions.length-1)
randomProduction = productions[randomIndex]
productions.delete(randomIndex)

// Make the new feature structure (that will be passed
   down) by unifying the passed down feature structure
   with the feature structure of the production rule

newFeatureStructure = unify(featureStructure,
randomProduction[1])

end while

// If the new feature structure was not able to be formed, but
   there was a child in the production, fail the algorithm

if newFeatureStructure == null and
   randomProduction[1] != null then

return null

end if

// Update the production counts by adding one to the
   production rule that was chosen

productionCounts[randomProduction[0]] += 1

// Recurse through the child symbols. If a failed route is
   found, fail the algorithm.

i = 0

// for every rule based on this symbol

for rule in randomProduction[0] do
```

```
// for non-terminals, recurse

if rule in cfg.productions then

// if the current feature is the head node

if randomProduction[2] == i then

featureStructureToPass = newFeatureStructure

end if

// if the recurse returned a failure, fail

if sentencePart == null then

return null

end if

// otherwise append the word part of the sentence

sentence += sentencePart[0]
oldFeatureStructure = newFeatureStructure
newFeatureStructure = unify()

// Make the new feature structure (that will be passed
   down) by unifying the passed up feature structure
   with the new feature structure formed earlier of
   the production rule

newFeatureStructure = unify(newFeatureStructure,
sentencePart[1])

// if the old featureStructure was not empty but the
   new featureStructure is now empty, fail

if oldFeatureStructure != null and
   newFeatureStructure == null then

return null

end if
```

```
//otherwise simply append the symbol to the sentence

else

sentence.append(rule[0])

end if

i += 1

end for

// return the current branch of the sentence and the feature
   structure that should be passed up

return [sentence, newFeatureStructure]

end if
```

Appendix C

Software Requirements Specification

C.1 Introduction

C.1.1 Purpose

This document gives a detailed description of the requirements for the *Biblical Greek Practicer* (BGP) software. It describes the purpose and requirements for the development of the system. It also gives the system constraints, interface and architectural design requirements, assumptions, and dependencies of the software. This document serves as a proposed plan to the user and guide to the developer.

C.1.2 Scope

Biblical Greek Practicer (BGP) will be a piece of software that will provide students with training sentences in the style of the Greek of the Bible. It will incorporate the vocabulary, grammar, and syntax rules used in the Bible and attempt to construct sentences that will work well for students to practice translating.

BGP will be designed in a way such that it will be extendable to other languages. Thus, the various modules of the program must be treated independently and have specific inputs and outputs.

C.1.3 Acronyms and Abbreviations

BGP – Biblical Greek Generator
CFG – Context Free Grammar
GUI – Graphical User Interface
NLG – Natural Language Generation
NLP – Natural Language Processing

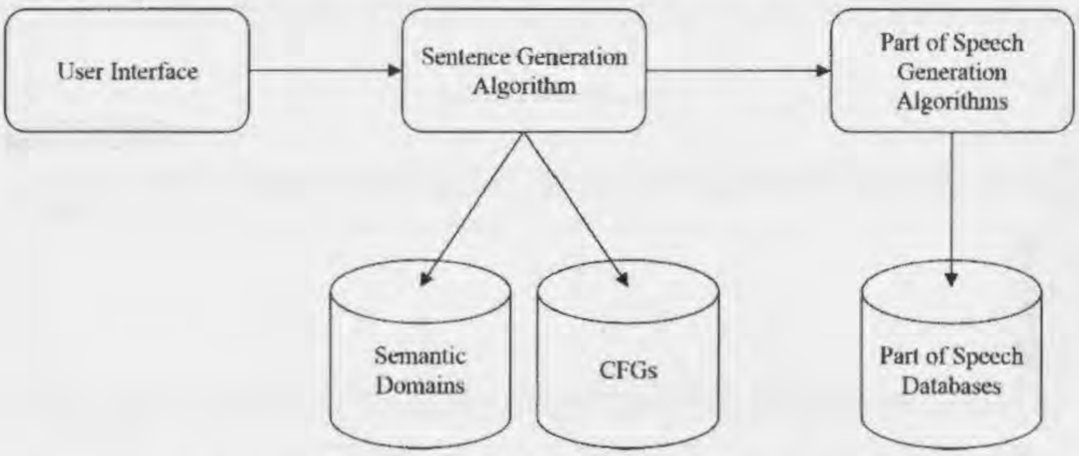
User Interface
Sentence Generation Algorithm
Part of Speech Generation Algorithms
Semantic Domains
CFGs
Part of Speech Databases

Figure C.1: General Process Diagram for BGP

C.1.4 Overview

The remainder of this document describes the complete requirements of BGP. Section two provides the overall description of these requirements, and is only meant to provide a general knowledge of these requirements. Section three provides the exhaustive technical details of the requirements.

C.2 Overall Description

C.2.1 Product Perspective

BGP will consist of a desktop application which generates and displays sentences to the user. On the user interface, the user will be able to trigger the generation process. When this is triggered, the sentence generation algorithm will communicate with two databases, one which contains the semantic domains of Greek words and one which contains the semantic-based context-free grammar of the New Testament. A sentence diagram is then constructed, and filled in with words by using part of speech generation algorithms, which must communicate with databases containing the parts of speech and conjugation rules for Greek words.

This software will be independent and self-contained. It will not need to access any external databases at present. However, because it is being written in a way such that it may be extendable to other languages, the system should be designed so that in future versions, it may communicate with other databases (internal or external) based on the language. Figure C.1 shows the general processes that will be required by the program.

Interfaces

The system will have several interfaces (see Figure 1). The first is between the user interface and the sentence generation algorithm. Depending on the language being generated, this main algorithm may have to change. This algorithm will in turn need to interface with two databases and a set of other algorithms. The two databases will contain the language specific CFG and semantic domains. The sentence generator algorithm will also interface with part of

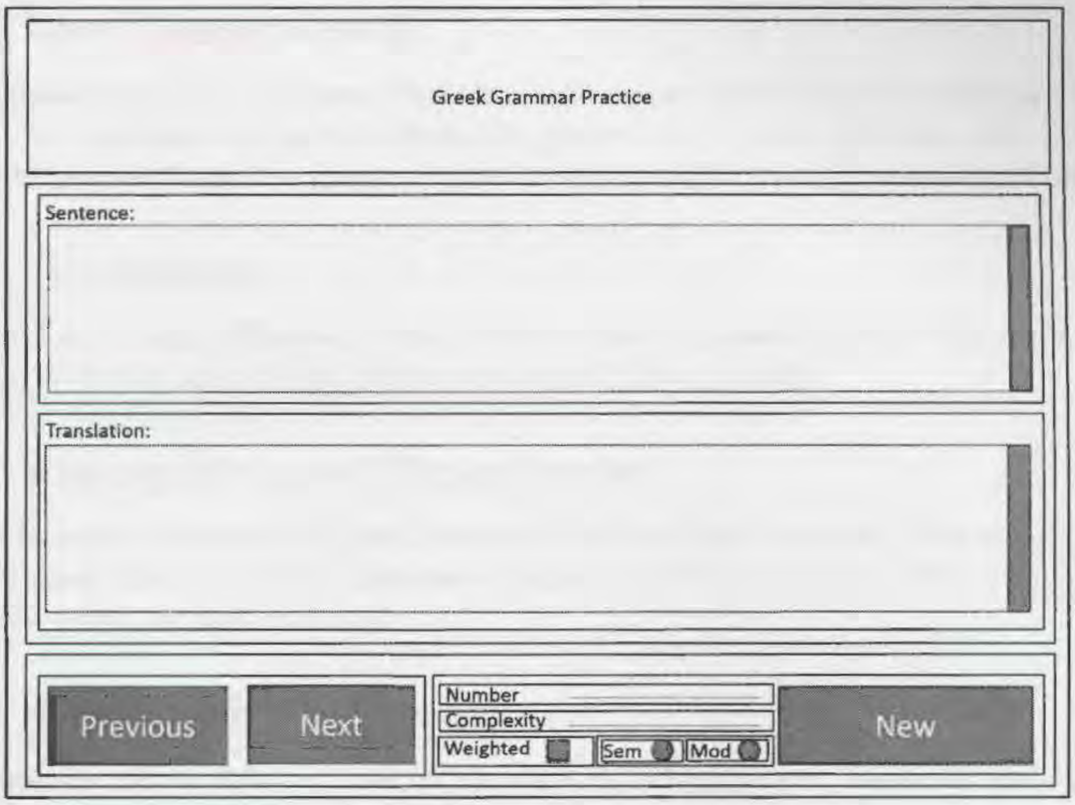

Figure C.2: GUI Diagram for BGP

speech generators, which conjugate words based on language specific rules. These algorithms interface with part of speech databases, which contain the language lexicons.

The user interface is the only means for the user to cause the system to generate sentences. It will consist of a GUI with a button that causes the system to generate and return a sentence to the GUI. The user will then be able to see the sentence. Figure C.2 shows an example layout of the GUI.

C.2.2 Product Functions

This section will give the user level functions of the system. These are general functions; the details of each are given in the third section of this document.

1. BGP shall provide a function to generate sentences in Greek.
2. BGP shall allow the user to specify the number of sentences to generate.
3. BGP shall allow the user to specify the complexity of sentences to generate.
4. BGP shall provide an area to display the sentences which are generated.
5. BGP shall provide an area in which users can type translations to the sentences.
6. BGP shall allow the user to navigate between sentences that are generated, and shall store user translations when switching sentences.

C.2.3 User Characteristics

The user base should be considered students and teachers with the least technical ability, so that the user interface can be built from the ground up to work with any user. Thus, the GUI should be built from the ground up to be a minimalistic and intuitive interface.

C.2.4 Constraints

BGP should work with Windows Vista, 7, and 8, and all versions of Mac OS X minimumly. Preferably, it should also run on recent versions of Linux as well.

C.2.5 Assumptions and Dependencies

BGP assumes that the user will have fonts installed on their computer that support Greek. It also assumes that all of the databases included with the system will be in place and untampered when the system is run.

C.2.6 Apportioning of Resources

Several features will be delayed until future versions of the system. Foremost among these is the inclusion of additional languages, which will include new databases that can work with the included algorithms. Also, more literature should be incorporated into the corpus, in order to improve the algorithms used. Other options should be included as well, including the ability to focus generation on specific grammatical structures (e.g. create sentences that include only the subjunctive mood), specific vocabulary words or frequencies (e.g. words used less than 100 times), and specific corpi (e.g. using CFGs generated from only Johannine literature). The system may incorporate multi-threading in the future, enabling faster generation. It should also allow the export of sentences and translation for later viewing and use in external applications.

C.3 Specific Requirements

C.3.1 Functions

This section gives the specific functional requirements of BGP. These requirements are more specific than the user level requirements of Section 2, as they outline detailed requirements of the system.

1. BGP shall provide a function to generate sentences in Greek.
    - 1.1. The function shall be activated by the user from the GUI.
    - 1.2. When the user activates the function, the system shall run the sentence generation algorithm.
    - 1.3. Based on the input given by the user (see below), the system shall generate the sentences via semantic-based generation and return them to the GUI.

2. BGP shall allow the user to specify the number of sentences to generate.
   2.1. On the GUI, the user shall be given the option to change the number of sentences to generate.
   2.2. There shall be a default number of sentences selected upon opening the program.
   2.3. When the sentence generation algorithm is triggered, this number will be given to decide how many senences to generate.
3. BGP shall allow the user to specify the complexity of sentences to generate.
   3.1. On the GUI, the user shall be given the option to change the complexity of sentences to generate.
   3.2. There shall be a default complexity selected upon opening the program.
   3.3. When the sentence generation algorithm is triggered, this number will be given to decide how complex the generated sentences are.
4. BGP shall provide an area to display the sentences which are generated.
   4.1. Upon completion of the sentence generation algorithm, the generated sentences shall be outputted to a list.
   4.2. The GUI shall display the current sentence to the user and provide methods to change the current sentence (see below).
5. BGP shall provide an area in which users can type translations of the sentences.
   5.1. A list of empy user translations shall created upon generating sentences, of the same length as the list of sentences.
   5.2. The user shall be able to type translations to the sentences generated.
6. BGP shall allow the user to navigate between sentences that are generated, and shall store user translations when switching sentences.
   6.1. The GUI shall contain features that allow the user to switch the current sentence and translation being displayed.
   6.2. When the current sentence is changed, the the translation shall be changed as well.
   6.3. When the translation is changed, anything the user has typed into the translation shall be stored.
   6.4. When the translation is changed, the new translation displays anything the user has previously typed into the translation.

C.3.2 Performance Requirements

- The system shall be able to generate up to thirty sentences in one call.
- The system shall be able to generate sentences up to length fifteen in complexity.
- The generation of sentences shall take no more than five seconds to complete.

C.3.3 Logical Database Requirements

There shall be three main types of databases in the system: Semantic Domains, Semantic-Based CFG, and Part of Speech Lexicons.

- Semantic Domains
  - These database will consist of the hierarchy of semantic domains.
  - It will list each domain and the domains which are subclasses of those domains.
- Semantic-Based CFG
  - This database will consist of the grammar of the language.
  - It will list the production rules of the language, based on phrase structures and semantic domains and conjugation forms (e.g. mood, person, or tense) allowed in each structure.
- Part of Speech Lexicons
  - These databases will consist of every word of a part of speech, and the rules which are used to conjugate the rule.
  - For each part of speech, will list each word and the conjugation rule.

C.3.4 Design Constraints

Standards Complience

The system should generate sentences that are grammatically correct. Grammatical rules shall be coded in and extensive testing shall be done to ensure that these rules generate correct sentences.

C.3.5 Software System Attributes

Reliability

The system shall provide methods that cancel generation should the process take too long or result in errors, in order to prevent crashes.

Availablity

As the user types, the system shall store translations in the temporary memory of the operating system, so that it can restore a user session should the system crash.

Security

The system shall conduct data-integrity checks upon starting the system, in order to ensure that data has not been manipulated by external processes.

Maintainablitiy

The system shall be designed in a way such that the editing of modules does not effect other parts of the system. Databases shall also be constructed in a way that supports easy editing in the form of additions, deletions, and manipulations of the data.

Portability

The system is to be designed as a client-side desktop application. However, it should be written in a way that it may be changed to a server-side program. This will enable the user interface to become either a web-based or mobile-based GUI, so that the system may be used more widely.

Appendix D

Test Suite

This appendix lists a suite of sentences output by our system. There are one hundred sentences listed for each sentence generation method, generated by calling a special function in our program to generate random sentences. They feature the same style of syntax, grammar, and vocabulary that can be expected by the system.

D.1 Modelled Sentence Generation Test Suite

1. ἔπειτα καθότι κέκληκεν ἀπέναντι τοσοῦτον Ἀαρών καθότι Ἀαρών ὡς κλῶν ὤόφείλεν ὑπόμνησεις

2. ὅς ζητῶν παρεκτός μοι εὐπροσωποῦν φιλοπρωτεύει καθότι οἷος εὐλόγηκεν ὁμοθυμαδόν πηλίκος ὀρθοποδοῦν

3. ὅς γάρ χαλκεύς μετά τῶν Μαθθάτ οὗ κυβείας

4. οὗ ἁπαλός δαπανάουσα Γεθσημανί ὀψέ αὐτόῦ τηλικοῦτης εγέμεν τό Ἠλί ἐκεῖνης ῥακά θεοσεβες ἑώρακυῖα ἑώρακεν ἀντί τούς ἅλατας οὗ

5. ἀφιλάγαθοι ὅταν οἳ Ἀπολλωνίαι διόπερ ὅς ἀγριέλαιος ἀνά Θεσσαλονικεύος τέταχασιν

6. ὁ καθό καταστροφή πεπαλαίωκεν τοιοῦτῳ ὅ μέντοι βδελύσσομαῃ ἀμάρτυρα ἄζυμα ᾧ ἄοῦντι

7. ἄδικος ὅς τετίμημενος κατενώπιον οὗ πινακίδιου Μαθθάτ ἀληθῶς ἀνθομολογέομαει καθώς Βοανηργές ἑαυτοῦ παρεκτός ἑαυτῷ φρεναπατάει

8. μεμένηκα ἥ Ἑβραϊστί Σαδώκ καίπερ τυφωνικόν ὅτε αἴγειον δεινῶς ἠσθένηκεν ἐντός τό ῥαββί αὐτόῦ

9. ἐντεῦθεν πρεσβεύοντες ἆρα κλίνομενοι κἀκεῖθεν ἠσθένηκοτες ἃς Γεθσημανί οἷς ἤγνικασιν

10. ὡσεί διόπερ ἥ Συχέμ ἃ ῥακά οὗ ἔριφου χορτάζῃ ἰδού ἥ Βοανηργές τοιοῦτου ἐπί Ἀαρών κεκολόβωται

11. ὥσπερ ἐντός ἤγνικαμεν ἔναντι ῥακά προφῆτιτα μέχρις ὑπερεκπερισσοῦ εὐτόνως πεζεύομεν

12. τότε εὐμετάδοτοι ἄχρι οἷ ὑπηρέται μεταξύ ζύμης ὅπου ὧν μήποτε τοσοῦτοι πέπτωκασιν ἐπάναγκες σείσουσιν ἃς Συχέμ αὐτας

13. δέδυκως ἵνα ὅς σωτήρ μεμίσηκεν αὐτῳ ἵνα ηὗλισμαως τετέλεχεν ἐγγύτερον πίμπλασθε ἤ οὐδέ Μαθθάτ αὐτοῦ ἐπάνω σεαυτῷ δουλαγωγέει

14. δυσκόλως οὗ τετύφλωκεναι προθύμως μακροθυμέονται ἄζυμοι ἤτοι μαθητεύουσιν ἵνα διάκρισεις εὐχαριστοῦσι Τιβεριάτος ἤ Βοανηργές σπίλοι κυροῦντες

15. ἡνίκα ἄφνω λελύπηκασιν ὦσθαι τοῦ Γεθσημανί κατενώπιον οὗ φάρμακου διότι πεπότικοτες μεταξύ ἡ ῥαββί σου ἔρρηγασιν

16. ὥσπερ ὅς ἔννοια γεγέννηκεν τοσοῦτῳ βραδύου ἤ Γεθσημανί σου πέπτωκεν ὅσον

17. μηκέτι ὑπέρογκον μηδέ τέ ὀρθῶς τοσοῦτοι τήν Σαβαώθ τοῦ Βοανηργές λυποῦντες ἐκεῖναι πόρρω οὗτοι πλησίον ἀλλήλοις αὐλίζομῶμεν Ἠλί πράσουμενοι ἣν ῥαββί τοῦ ῥαββί ἀλλήλων

18. ὁ μέντοι δύναμις ἤρκως τόν Ζεύς σέσηπεν πάντοτε ἔκράζεν ἀλλά ἀντί ἄλλων συνηλικιώτῶν στρατολογῶν νενίκηκεναι τοῦτον ἔμπροσθεν ὅ γράοῦν κατά ποῖου καί εἠγεμονεύεν τι ῥαββί ἤθληκεναι ἐπέκεινα σεαυτοῦ πλατύουμενον

19. ἤ κἀκεῖ Σαβαώθ κέκομται χυκλεοῦν

20. τῇ ἐπεί πέριξ ἑώρακοτες λέλοιπαμεν ὑπό Καῦδα ὅθεν γεγήρακοτες ἐγγύς ὃν κίνδυνον θώρακος τοῦ Εὔτυχου θυμιοῦντος ἀπό ὧν ἐννέα πέπτωκαμεν ἄχρι ἑαυτῷ

21. Βηθσαϊδά πλήν ὑγιει ἤντληκαμεν ἱνατί κακολογουμενοι

22. μέντοι εὐλόγηκα τόν μνᾶν τέθυμενον γάρ μήτι ὄφελος ψυχικός μή ὁ ἰάομωμενος μετά ἐκεῖνον βρώσιμος θεμελιουμενος ὅθεν ταχύς τοίνυν ἔξω Βοανηργές ἀσεβέει ὡς ἀγραυλέει

23. κἀκεῖ ἠσθένηκεν Ἠλί αὐτόῖς

24. μέν ἄτομοι ποῖου φεύγομεν

25. Καῦδα κἀκεῖθεν ἐβδέλυγμενης πεπίστευκεν ὅς εὐπορία τῇ Βοανηργές μοι ἐάνπερ δωρεάν πέφρικεν ἤθληκεναι τινα πρός ὅ καλῶς εἴληῶσθαι ταυτην

26. ὁ ὅτε ἤλπικως τό Μαθθάτ μου ἤτηκως ὑστέρηκεν ὑπέρ ὃν βόρβορον

27. μᾶλλον πρεσβύτερο μή μή εὐσεβῶς οἱ καρδιογνώσται ἐκεῖνου ὁμοιόονται διόπερ θρόνοι Καῦδα αὐτῶν ὅ Συχέμ πεφίμωται κατέναντι ἃ Ἀαρών ὧν

28. ἤ Μαθθάτ οὗ αἴνεσεως τινων πρόσκομματος ὕλης ὀψέ εὐπρόσδεκτων ὧν

29. ἡ ἤπερ πειράουση εφάσκεν ὅς ἀφειδία ὀψέ ἐκεῖνοις ἄχρις δοκιμασίαν βίαιοι δέ πεποίηνται οἳ εὐπρόσδεκτοι

30. τά ὡς μονογενή ἐκεῖνου πρός ῥακά πρεσβύτου οἷς Καῦδα ψηφίουμενα διστάζεται ἤ ὅταν ἄτεκνος ἐκεῖνου Καῦδα μέχρις Γεθσημανί ὑποκάτω ὅ σπαταλοῦν ἀλλήλους δῆλους

31. βέβληκα ὅθεν ἠγάπηκεναι ῥαφίτα παρά τοῦ Μαθθίου ποῖου ἵνα Συχέμ ὑπερέκεινα ἢ ῥακά ταυτης κἀκεῖθεν ῥαββί ἄντικρυς τῆς Σαβαώθ σου ἤ κραταιόι οὗ γνώμης οἱ πλοες μου

32. δέ δεδεκάτωκα κἀκεῖ Ἀαρών ἀπειθή πεπόρευμενην ῥαββί ἕως τούς ἀσεβοῦντας ὀψέ οὗ Σαβαώθ καθά χωρίς ἢ Συχέμ τινος καίπερ μέχρις τοῦ ἀπόδειξεως οὗ Ἀαρών μου λέλοιποτας ἐπέκεινα ἥν Ἠλί τήν κριτικην κακοπαθοῦντας Συχέμ οὗ Κλαύδιου

33. τόν ἀπόκρισιν ὅσον ῥαπίσει τοῖς ἀνάγαιοις ἀναβολής πιστεουμενος ῥακά μετά ἱεράτευματος τρύβλιου τοιοῦτος λικμάει ἄτομων Ἐπικούρειος αὖτα πέπτωκατε ὅ πέπραμενον Ἠλί μέχρις σύντροφου τῆς ῥαββί ἠμφίεσμενος ὑπέρ ἢ Σαβαώθ ἐπέκεινα τό Γεθσημανί πόσον ὀμώμοκεν ματαιότης αὐτόπτην τόν σύν Καῦδα ὡσεί τέτιλκεν τοσοῦτον ὅς πολίτευμα Συχέμ λιπαρῳ ἕως Ἠλί πηλίκος ὑστέρηκεν ἐμῶν καίπερ πνίουμενος ἄκρατοιυς τούς στρατολογουμενους ἐντός οὗ χρυσοδακτύλιου πρίν ὅς κράτος ἐμοιχεύεν εἵνεκεν τηλικοῦτου

34. λάθρᾳ γεγάμηκατε κἀκεῖ ὄνοματα ἥρπακαμεν δίς διάφορα

35. καθάπερ κέκτηται ἄτερ ᾧ βαστάοῦν σε βασιλεουμενον ἀπό πυκνας ε;εὖχομουν ἐκεῖνῳ οἱ κατάβασεις ἄρα ἐνήνοχεν ἐκεῖνους πυνθάνομῶν ἐμαυτο ὅσο οἱ ἤθη κακοποιέουσιν κλεοῦν

36. ἄκανθαῶν ἀγνοέοῦσθε ἐπάνω αὐτῶν

37. γάρ τέθραυσμενος ὁ συνοδία ἧαἰνα σοι εἴτε οἷ ψῦχη τοιοῦτου

38. πρίν εὐλόγηκεν ποῖῳ ὁ στέγη πῶς πέφευγεν ἄντικρυς τινος τό Σαβαώθ

39. εἴπερ ἡγίασμενος Ἠλί πεπίστευκως λελύπηκεν ἥρπακετωσαν με μήτε τέτιλκετωσαν πρός ἐκεῖνους

40. οἷ ὅταν Τίμωνες κέκλικοτες με πέπτωκεν δῶρον ζώσουσιν τήν ῥαββί ἑαυτῶν φρονίμως σαλεοῦντες οἱ ἔντευξεις δεδυνάμωμενοι ἡδοναῖς νενόηκασιν ἀπέναντι ποῖα

41. ῥαββί αὐταῖς ἧπερ Σαβαώθ παρά ὁ νοῶν μήποτε ὅς ἐλιθοβολέεν οὗ ὁ μακροθυμουμενος μέχρις εἰς τῶν ὀκτώ Καῦδα τινα ἐνώπιον οὗ μάγου ἑαυτοῦ οὗ ὡς κατάνυξεως Δανιήλ ὅς ἀπώλεια ὅς συνετός ὅς σκολιός ὧν ὁμόφρων καί ὁ πρωτοτόκια τῶν ἔλεων τῆς ῥακά

42. ἄχρι λέλυται εἴπερ πεπλήρωκεν τόν σῖτον ἢ ῥακά ἡ ῥαββί κεκάθικεν τό Σαβαώθ ἀπό τῇ Συχέμ αὖτης ἕως ἔσκυλται Συχέμ θεομάχου ἤ Συχέμ ἐπειδήπερ ὑστέρηκεν Σαβαώθ ἔρημῷ ὅτε ἠγάπηκεν σεσάρωμενη οὗτος κατά Γεθσημανί μέντοι ἤγγελμενος ὁ Ἀβιούδ τῆς Συχέμ οὗ

43. ἔσχηκατε ἀλλά τούς Βαράκ ποταπούς διό κόκκινους καίτοι ἱλάσκομῶντας τήν Σαβαώθ πόσων

44. ἐάν καίτοι Ἰωδά ἵνα δεδάμασται εὐχάριστον καθότι ὅ Ἀαρών τινων πενιχρα συντόμως ἤ Ἠλί τοιοῦτων λογίζομαομεθα δέ ἀναντιρρήτως ἀπόκριματα τοῦ πόρνης ἐπειδή στηρίξουμεν παρά οὗ λίνου ἵνα κεχώρηκεν ὃν μέτωπον τοῦτον μήτιγε ἠγάπηκεν καθά καθά ἄπταιστοι ἀποτόμως στερεόονται

45. ἄρα αὐτός ἅπτει ἔσω ἀλλήλων Φορτουνᾶτος ὅσον βέβληκεν ὁ Ἰωσῆς αὐτόῦ ἱεράτευμα οὐχί λόγον δέδειχεν τινι οὗ ὄντως ἀνθρακιάν ἠκολούθηκεν οὗ οἶνον κεχώρηκεν αὐτῳ

46. ἕως ἐγώ ἴαμα ὀργίλος ἀντιπέρα Μαθθάτ Ἀαρών μου φωνῶν ἔτίλλετο ἐμαυτο ἐδοκιμάζον ὄπισθεν Μαθθάτ ἔξω τήν Γεθσημανί οὗ ἔκφοβου ἥν πλήσουμενην σιτευτην οὗ ἀκοοῦν ῥαββί ὑπέρ τῶν τρέουμενων ἔξω τό ἀσφαλες ἐκεῖνος πέπραχως ῥύπον μήτε κλισίαν θρηνοῦντας πατοῦν ἐντός ὅ περίχωρον ἐδοκιμάζεν Ἠλί δέδειχεναι

47. πρίν δόλιον Γεθσημανί ἄχρις ἣ ῥακά σέσηπεν

48. κεκάθικως πρίν ταδε ὅτι κέκραγως μολυσμόν πέποιθεν ᾧ ἐλαίᾳ ἀντιπέρα ἀντίτυπων ἐπάν λατρεύων κεχειροτόνηται ἀρκοῦν

49. τινα ποταπη δέδειχα ὑπό ᾧ τελευτη κίχρημι

50. ἤἐγείρον καθά τῷ ἀντάλλαγματι μου ὅ ὅσο γέ φορέοι ψάλοῦσθαι ἐκεῖνο

51. μεμαρτύρηκας αὐτόν τυπικῶς ὅ κατενώπιον ταφάς Γεθσημανί ὅτε Συχέμ μελλήσεις τοιοῦτον μηδείντα πέπραχας ἐκτός ὧν λιμήνων μου

52. ἐπάν τοιοῦτος τύπτει γῆρας κληροῦν ἐπειδήπερ συνετός σκιρτου ὅσα ἔοικα αἷς μήποτε τεκμήριου σωφρονίζει

53. ὅς φραγελλόεται πρός ἑαυτῷ ἀντίδικῳ ἁδρότηπι τοσοῦτῳ ἐπηρεάζει Καῦδα εἰς Μαθθάτ

54. ὁ χοῖνιξ ἑαυτῶν πάλαι ἐπάνω Βοανηργές Ἠλί ἔσκυλμενος κεκληρονόμηκεναι μάτην φυλακίζει τινας ἀνέγκλητῷ φίλανδρῷ τρυγοῦσθαι

55. οἱ κἄν κολακείαι στόρηοντες νενόηκασιν ᾧ

56. κἀκεῖ κέκρικως κεκοίνωται Ἀαρών τισιν οἷς ἄμαχον ἐάν Συχέμ οἷς ἀκαίρως πλήν ἐκτός μου φωνέομεν τήν Συχέμ οἱ ἀνόσιοι ὄπισθεν ἔνατῳ Μαθθάτ ἐπί ὃν φωστήρα

57. εὖ χαρίζομαι ἥν Ἀαρών τοῦ κόλπου

58. τετέλεσμενος διό αὐτόυς σώσει

59. ὁ καθώσπερ ἠδίκηκεν τοίνυν ἐσκανδάλισται ἡ ῥαββί ἑαυτοῦ

60. ὁπότε ἐταλαιπωρέεν χωρίς ποῖο ἔγνωκα τισιν δέ ἄπταιστος χαριτόεται ἐνήνοχεναι ἐκ οἷον καί ὀψέ βλασφημέη πεφυσίωμενον τοσοῦτῳ ἄχρι οὗ κόκκου

61. εἴληφως πρίν ὅς ἀντάλλαγμα πεποίηκεν αὐτῳ κράουτω πόσῷ ποῖο φιλήδονος πυνθάνομαεις

62. ἀλληλουϊά μή δημηγορέω ἑαυτοῖς τρίς δέρει φάτνη ἰσότιμος τοῦ Ναγγαί ἐμαυτοῦ διό χαλκός συστατικός οὗ ἤλπικοτος τοῦτον

63. ὅπου πεφόρηκοτες οἱ μετάλημψεις ἄχρι ὅ ἀναξίως μεμίσηνται Φρυγίας ἤγνικεναι

64. πελεκίζομαι ἄρα τοσοῦτῳ προσωπολήμπτη δείξεσθα σε

65. σέσωκεν ἐάνπερ ὅ Γεθσημανί τῷ φιλόσοφῳ ἔγνωκε ἐπειδήπερ χρημάτιηθι ᾧ Συχέμ μοι

66. κατωτέρω ἀδιάκριτον τό Σαβαώθ ὧν

67. Ἄζωτος τετήρηκεν ἐκπλήρωσιν κἄν γεγένηται τοιοῦτον κονιορτόις ὅτι ἐλήλεγμαεν δέησεις πρόθυμοιυς

68. δέ ὁ ψευδώνυμος ἔρριμμενος κοινωνέῃ τήν τουτου ῥακά ἐπάνω Συχέμ ἀρνέομαει ἃ ἅλ-λομῶντα ἐκεῖνου

69. οὐ ὀψέ σάρκα τοιοῦτα στοιχέω μέχρις ἵνα ὅς μέθυσος ἑαυτα δή θριαμβεύει

70. κέκληκεν ἑαυτοῖς σικάριος δή μενοῦνγε ὀχλοποιέω αὐταῖς γάρ ᾠδήν ὦσθαι οἷα δουλαγ-ωγέω

71. τοσοῦτος ἵνα ἔρρηγεν ὃν μέμιγται οὗ αὐτός ἠσθένηκεν τοσοῦτον ἐσπαργάνωται

72. μηδέ κορέννυασιν μοι βασιλεύει τι ἀποτόμως ἐγγίωσειν ἀντιπέρα κοπετόις οὖν Σαβαώθ καθά ῥακά εἶτε σορόις παραθαλάσσιοις

73. πρίν ἔμπροσθεν σύ μεμέστωμαι ἤγγελκως ἔμπροσθεν μοι

74. δεκτόν ὅσο πήγνυμαι ἔναντι τοσοῦτους

75. ὀλιγωρουντων ὁσάκις ἐκεῖνων δεδώρηται πλησίον ἐκεῖνον Ἠλί οὐδέ βούλομαει τόν Ἄσσον ἤγγελμενον ὥσπερ ποιοῦν ῥαββί ἐκεῖνο ὡς ῥαββί αὐτόματην γραπταις Ἠλί ἀριθμουμενον ἀντί ἢ Ἠλί ἀντιπέρα τουτῳ ἐκομίζεν κωφα ἃ ὀλίγα μέν Σαβαώθ τῆς Μαθθάτ κἄν ἀμετάθετα τοῦ Κυρήνιου

76. ποτέ ἀμάω κυκλόθεν ὁποῖαν ἄρρωστοε γραπτε ἀκμάζουσθω τοιοῦταις τοιοῦται ἐπάνω τῷ ῥακά τετελεύτηκασιν μοι πέραν διότι ἀσπίτος ὥστε τῶν θειώδων καθημερινων σου τοσοῦτων ἃ Σαβαώθ κύκλῳ Βοανηργές ῥαββί

77. δέδυκεν εἰς τόν στάμνον ἔζωσμαθω τρίτον ἤτοι ὑπηρετέει οἷον

78. καθά εἴπερ ἀξίως τετήρηκας ἐκεῖνο θεωρέῃ μή πεζῇ ἠγάπηκως

79. κατέναντι αὐτῷ ἀκμάζομεν ἄρα ὑποκάτω ἑαυτῷ μωμάομαομεν ὅτε οὗτος μετά οἷς ἐάν ἐναντίον οὗ Καῦδα ἐμαυτοῦ δεδούλευκεν οἷς

80. ἄχρι ὅς στολή ἐκεῖνον στερεόετε ἔξω κελεύει σκυθρωπός ὅθεν τοῦ θαυμάζοντος τινα κυβέρνησεως

81. ῥαββί διότι ἠγάπημενης ἑώρακεν Ἠλίαν ὁ Ἀχαϊκός μεσημβρίαν σχῆμα ὀλολύῶν διότι ῥαββί ὦσθαι οἷς πολυτελεσιν ὁ διαμερισμός λελάληκεν ὃν Μαναήν κέχρημενον

82. μόνον εὐχαριστοῦσι με

83. ἠπόρηκεν κἀκεῖ δεῦρο διχοστασία ὁ λέλοιπως ὀψέ ὅσον Συχέμ τό ἀμετανόητον μανθάῶν Σαβαώθ Ἠλί ἐπειδή ῥακά ταχέως Μαθθάτ τεσσεράκοντα

84. ἐμοιχεύον ἐπεί ὄπισθεν τοιοῦτοις τεσσεράκοντα Ματθάν

85. δέ ὀμώμοκεν ἡ Ἠλί τῇ Βοανηργές ὅταν πέπομφεν ἡ Ἠλί τό Ἀαρών ἐκεῖνης ἐπεί μεμαρτύρηκεν ὃν ἵδρώτα ἑαυτον τετελείωκεν ὁ γόνυ ἔξω τοῦ Γεθσημανί οὗ

86. ὁ πρίν ἡγίασται ζυμῶν λάθρᾳ νενίκηκα σε Σαβαώθ

87. ἔσύρον πλήν οἳ μεγαλοπρεπεις ᾧ δεδούλωμενῳ Συχέμ μοσχοποιέει γάρ κακῶς ἐρείδει αὐτῳ βέβληκεναι τόν θήραν

88. μή εδιχοτομέεν οἷς ἐκεῖναι ὑπερέκεινα ὧν Ἑβραϊστί γαμέετε ἐγώ ἔσω ὧν ἔτι φημί αὐτόι ἕως οὗ οὗ ἐνδώμησεως σείετε τοσοῦτος ποσάκις μάχομαι ἐγγύς οὗ κατήγορου αὐτόῦ

89. φυσικῶς ἀφιλάργυρον καθάπερ ἕνεκα τῇ Συχέμ ποῖου τοιγαροῦν ἡδέως ἀνά ᾗ Ἠλί τινι λέλυται πλησίον ἐκεῖναις τελευτῶν ἀλλήλοις τήν ἑαυτῶν Μαθθάτ δν ἑαυτῶν πτόησιν δν τινων συγγνώμην ἅμα τουτου πῶς ἦν ὀψέ ὦσθαι μήτε μηδέ ὁμοίως μεμαρτύρηκα ἐκεῖνας ὑποκάτω ᾗ Μαθθάτ Ἰουδαϊκῶς βαστάζομαι ὡσπερεί καθεξῆς ἐκαρποφορέομην πολιτεύομαι εἰ ἡ Ἀαρών τοιοῦτη πλήν ἑκάστοτε κατά Σαβαώθ κεκληρονόμηκεν πόσους πού εὑρίσῶ πρωΐ μέχρις ἔσπαρσθε ὡς ὥσπερ λελατόμησθε ἐκτός ῥακά

90. δεῦρο κοινωνέει οἰκτίρμων ὁ ἐσπαργάνωμενος ἔσω οὗ ῥαββί

91. ἥτις διόπερ καθώς γεγήρακεν κεχρημάτισται ὕψιστον τοίνυν ἐστενάζετο μέχρις τοιοῦτον

92. κεχοίνωκως ὅθεν ἔκτος ὡς ὧν Μαθθάτ συκάμινος σέσωκεν ἐκτός οὗ Καῦδα ῥαββί Ναζαρηνην νουθετοῦν ἔστασθαι ὀψέ σύμφυτον ἥν Καῦδα ἐκεῖνη κεκάθαρται πλήν Ἐμμανουήλ

93. πατροπαράδοτου δέ λελατόμημενης μεμέρικασιν μοι ὀλιγωρουμενους ὀξεις

94. ἥρπακετε ὀπίσω ᾗ Βοανηργές τῇ εὐλαβει

95. κεχώρηκεν διόπερ κατά ἀλλήλους οἳ οἱ ἄψινθοι τινες ὅσους ἐλευθερόετε ἐπάνω ἐκεῖνους καυχάομῶντες

96. σφοδρῶς πλήν ὅς ἐμπαιγμός μηδαμῶς σοι ἤθληκεν

97. μήποτε ὅς ὀλιγωρέῃ ὅ Μαθθάτ τινος γεννοῦν ἐγήγερται πέραν ᾗ Γεθσημανί ἵνα ἔμπροσθεν τοῦ περίψηματος τίκτει οὐδέ ἡλίκον ὀπίσω οὗ βαπτίζω

98. ποῖο ἠπόρηκεν κεχώρηκεν

99. εἴτε κέκλικεν τοιοῦτῳ οἰκετεία ἄτερ Σαβαώθ ἐρωτάεται τοῦτο εὐπρόσδεκτον μωμάομᾶν

100. πλήν τοιοῦτο γεγένημεθα

D.2 Semantic Sentence Generation Test Suite

1. γάρ μή μή ἀλλά ἀλλά λοιπόις πειρασμόις

2. ὅτι σήμερον γίνομαομεν μετά ἑνδέκατος ἀριθμόις ἤ θρίξιν

3. οὐ ἕως μήτι ὥστε καί ἐν χιώνων ὀρύσσουσιν

4. καί εἱμεσθω σε ἔνταλμα ὦ Οὔ Ναί μενοῦν

5. τοίνυν πῶς ἡ τίς κἄν ἤ
6. καί οὔτε ἡ ἔξω ἄνωθεν αὐλίζομἄν
7. ὅτι μήγε μήτι δέ πλήν τελευτή
8. διά αἳ διετής σῶματα βέβληκεναι αἳ πλησίον
9. τέ ὠδίνι γίνομαεται μονογενει ἐγίνομαομην ὧδε
10. ὅτι οὐαί πανταχοῦ πῆχυν καί ὅθεν
11. μηδέ εὐλόγηκεν ἡ ἐκ τινι δέδεκεν
12. μήτε ἤγγικοτα μηδέ γεγένηται ὅπου ἤ δίκτυα
13. οὕτως τρίς οὐδέ ἑβδομηκοντάκις ἑκατονταπλασίων ἄχρι δίς
14. ἰδού ὅπου καί πάντοθεν οὐαί πόρρωθεν
15. ἐγγύς οὐδείσαν κωχαρίτωται ἔτι ἐγίνομαον ἐθανατόον
16. κἀκεῖθεν ὀρθρινός γίνομαου μόνος ἦν κἀκεῖ
17. ὅτι ὂν ὅταν ἐμεριμνάομην δέ θνήσκῃ ὅλον ἐκεῖνον
18. ὥστε ὅταν ἠεἰουμεθα καθώς τρίς κἄν ταχειαι μήτραι
19. μήποτε δή ὅς ἀντί ἐγώ νεότης ὄγδοος
20. ὡσαύτως ἐγίνομαεν εἲς ὕστερῳ συντέλειᾳ εἲς ὅταν
21. ὁμοίως ὄγδοος ἐκ δίς τέ κατωτέρω
22. γάρ μή οὔτε μηδέ μηδέ ἅπασαι ῥύσεις
23. ὅτι πάλαι εἶμομενην τρίτον πενταχισχίλιοι καί ὀδούτα
24. γάρ μή ἀλλά δέ τινων ῥάπισματων
25. δέ κατά ἱμάτια ἐκαθίζοντο οὔτε τρέουτωσαν
26. κἄν μόνον πολύν ὅ ἀδικέης τέ τάσσωσιν
27. οὕτως τέταρτον ἄχρι ἀριθμός τέ ἀνά δύο
28. ὅτι μήτε οὐδέποτε πλήν πλήν ἅπαση ὄνατι
29. οὐ ἐστίν ἥκει πρός ξύλῳ μήποτε ἄροτρῳ
30. δέ ἕτεροι τινες αὐλίζομἄν ἤ ἐγήγερται
31. δέ σύσσημα ὦσθαι μονογενεις γίνομαετω κἀκεῖ

32. κἀκεῖθεν πῶς ἡδέως ἐπέμφθησεσθα αἳ ὑμεῖς
33. καί ἃς γέγονεναι μόνας γεγένῆσθαι ὅπου
34. γάρ μήποτε οὐχί μήποτε ἐγίνομαομην Ναιμάν
35. κἀκεῖ μέσην ὦσθαι μονογενή γέγονεν ὅθεν
36. ὅτι γεγένηται ἑπτάκις ἤ δέκα καιρός φόνος
37. γάρ μή οὐ μήποτε δέ ἐκεῖνη ἀπαρτισμός
38. ἴδε τό φρόνιμον ὅπου ἰδού ὅπου
39. οὕτως δίς ἕως τρίτον οὐδέ οὕτως
40. μηκέτι κέχριται μηδέ ᾧ βολη κέχριται
41. μηδέ ὥστε κεκολόβωνται ὅσοι ἕνεκα οὕτως ἐκολοβώθησονται
42. οὕτως Ἑπτά τρεῖς ἐπί τῷ ἔνατῳ μηδέ πικρῶς
43. καί ἡ ὅλη γεγένῆσθαι μόνη εἰμί μακρόθεν
44. ὡσαύτως δίς πεντεκαιδέκατος διά τῶν ἕκτων οὐδέ μέχρις ἔνατος δύο
45. ἰδού ὅθεν οὐαί τά ἐλάχιστα ἐντεῦθεν
46. κἀκεῖθεν γεγένησαι ἀνά δίς τοῦ κοράσιου
47. μήτε γίνομᾶν ἤοῦν ἅμα διπλοῦς γάμος καί πληροῦν
48. ὁμοίως ὅ ἑνδέκατον ἀνά τρίς τέταρτος
49. μήποτε δέ με αὐλίζομαεσθω μηδέ φύομαι
50. τέ τῆς ἄλλης σέσωκετω οὐδέ πατοῦν
51. ἤ ἦν ἔτι πεντήκοντα πεντήκοντα μήτε χρυφαῖᾳ χοῖρῳ
52. ὁμοίως μηδέ ᾧ τρίτος ἕν ἅμα πενταχισχίλιοι
53. μηδέ ἅπασιν τουτοις γίνομαετε σάββατοις τρίτον
54. γάρ οὐδέ οὐδαμῶς ἤ δέ τελείωσεως ἐκεῖνου
55. κἀκεῖθεν οὐ κλαουτω καί ἐγγύς ὅ
56. ὅτι μήποτε μηδέ μήποτε ὁρκίου Ἰωσίας
57. γάρ μή ἤ κἄν οὒς λαῖλαπας
58. οὐαί ὄπισθεν μηδέ ὅπου ἴδε ὅθεν

59. μηδέ περίχωρος ἦν ἔνθεν ὅτε ἐγήγερμαθε
60. οὐ ἦν λεμά ἐλεύθεραν ἑαυτην εἴρηκεν γονεύν
61. τέ τέ ἅπασας τέτευχεναι μονογενεις ἔσομαονται ὅθεν
62. μήποτε οὐχί οὔτε πεπαλαίωκεναι ἐνιαυτόυ δέκα
63. τέ γεγένημενος ἔτι ἔνατος τέ δεκατέσσαρες ἕνδεκα
64. ἑβδομηκοντάκις ἑβδομήκοντα ἀλέκτωρος πεπαίδευμαι δέ βλέπητε
65. γάρ μή ἀλλά μήποτε πᾶντος προσευχῆς
66. οὕτως δίς ἥμισεσιν περί τρίτον εἷς μήτε μόγις
67. γάρ μηκέτι ἤ κἄν ὑμεῖς γάμοι
68. ὡσαύτως τρεῖς τρεῖς ἐκ αἳ ἑκατονταπλασίωνες
69. γάρ οὐ μή μήποτε οὐδέ ποικίλος αἶνος
70. δέ Ἱερουσαλήμ ἄν ἐγήγερται τέ Δεῦτε
71. ὡσαύτως τρίς σύν οἳ ὀγδοήκοντα τέ δεινῶς
72. οὐδέ μηδέ οὐχί ἐστήριγσθε ἑκατόν νεότηπων
73. ὅτι μή οὐχί δέ ἀλλά ἀπαρτισμόι
74. μηδέ ἦν ἔφθάνεν κἀκεῖ οὐδέ μάχαιρη
75. καί τυγχάουμενα ἅπαντα δεῦτε οὔτε Ὧδε οὐδέ ἐκτός
76. ἵνα ἐξάπινα ἔσομαονται ἕως τρίτον τρίς ἤ γλῶσσι
77. καί πᾶντα ἐπί τουτους γεγένῆσθαι μόνα γίνομαεσθωσαν ὅπου
78. οὕτως δεύτερος εἴκοσιν ἐν οἳ τρίς
79. μηδέ οἳ ἀρχαῖοι αὐλίζομαεσθε οὔτε ἠγέρθηση
80. ὅτι γεγένηται παραχρῆμα τῷ κλῆρῳ ῥύσει
81. πλήν οὔτε μήτε ἐστηρίχθησονται οὐδέ ὅσαι ἑσπέραι
82. ὡσαύτως γεγένηνται καί εἰς νύπων δίς
83. ὡσαύτως γεγένηται κατά αἰώνα ἄχρι πάλαι
84. γάρ οὐδέ μή ἀλλά γίνομαεται Ἀμώς
85. ἄν τέ ἐν τηνδε ἐστίν κύπτει

86. οὕτως ἕκτην ἐπί τρίς δέ σφόδρα

87. οὔτε μή ἅμα κόλασεις ὅταν εἷμομεναι

88. οὐ οὗ ἠἀναγκάζεν ἐάν πῶς ἄτεκνον

89. μήποτε οὐ εἵνεκεν πρᾶξει ἀλλά εἰεῖται συντέλειᾳ τρίς τρίτον

90. γάρ οὐχί μήγε κἄν ἤ λύτρων

91. κἀκεῖ ἤεἰουμην ἀπό οὗ τρίτος τριάκοντα

92. κἀκεῖθεν ἔσομαομαι τῆς ἴδιας ὀργής δέ ἐνθύμησεως

93. κατά γένεσεις κέκριχασιν μήτε μηκέτι κρινοῦσιν κέκριχασιν

94. ὅτι πρωΐ γίνομωμενην ἑνδέκατην οὐδέ δάκτυλον

95. οἱ λύχνοι ἐντεῦθεν ὦσιν γέγονατε οὖῶτα

96. ἵνα ἐγγύς γεγένηνται ἕνα ἐννέα οὐδέ αἷματων

97. οὕτως τρίτον δεκατέσσαρες περί ὅ τρίς ἕξ

98. ὅτι οὐ μήτι δέ κἄν λαῖλαπος οὔτε μάστιγος

99. καί ὅλην ἐκεῖνην γίνομᾶν μόνην γίνομαετε πάντοθεν

100. γεγένησθε κέκομσθε ἀριθμός οὐδέ τέσσαρες στρουθία

Bibliography

[1] *The Mnemosyne Project.* The Mnemosyne Project, 2015.

[2] *Rosetta Stone.* Rosetta Stone Ltd, 2015.

[3] *SuperMemo.* SuperMemo World, 2015.

[4] BIRD, S., KLEIN, E., AND LOPER, E. *Natural Language Processing with Python.* O'Reilly Media, Beijing, 2009.

[5] BLEDSOE, W. W., AND BROWNING, I. *Pattern Recognition and Reading by Machine.*

[6] CHOMSKY, N. Three models for the description of language. *IRE Transactions on Information Theory 2*, 3, '113–124', year =.

[7] COLLINS, M. *Natural Language Processing.* Columbia University, 2014.

[8] ELMES, D. *Anki.* 2015.

[9] GOLDBERG, E., DRIEDGER, N., AND KITTREDGE, R. Using natural language processing to predict weather forecasts. *IEEE Expert 9*, 2 (1994), 45–53.

[10] HARRIS, Z. S. *String Analysis of Sentence Structure.* Mouton, the Hague, 1962.

[11] HOBBS, J. R. Resolving pronoun references. *Lingua 44* (1978), 311–338.

[12] JURAFSKY, D., AND MANNING, C. *Natural Language Processing.* Stanford University, 2014.

[13] JURAFSKY, D., AND MARTIN, J. H. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition,* 2nd ed. Prentice Hall, Upper Saddle River, NJ, 2008.

[14] KLEENE, S. C. Representation of events in nerve nets and finite automata. In *Automata Studies*, P. A. Griffiths, J. N. Mather, and E. M. Stein, Eds. Princeton University Press, 1956.

[15] LITMAN, D. J., AND SILLIMAN, S. ITSPOKE: An intelligent tutoring spoken dialogue system. *HLT-NAACL-04* (2004).

[16] LOGOS BIBLE SOFTWARE. *SBL Greek New Testament Text and Apparatus.* Logos Bible Software, 2014.

[17] LOUW, J. P., AND NIDA, E. A. *Greek-English Lexicon of the New Testament: Based on Semantic Domains.* United Bible Society, London, 1999.

[18] MARCUS, M. P., MARCINKIEWICZ, M. A., AND SANTORINI, B. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics 19*, 2 (1993), 313–330.

[19] MARIA MILOSAVLJEVIC, ADRIAN TULLOCH, R. D. Text generation in a dynamic hypertext environment. In *Proceedings of the 19th Austalasian Computer Science Conference* (Melbourne, Australia, 1996), pp. 417–426.

[20] MCCULLOCH, W. S., AND PITTS, W. H. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics 5* (1943), 115–133.

[21] MORPHGNT. *MorphGNT SBLGNT.* MorphGNT, 2014.

[22] MOSTELLER, F., AND WALLACE, D. L. *Inference and Disputed Authorship: The Federalist.* Springer-Verlag, 1964.

[23] MOUNCE, W. D. *The Morphology of Biblical Greek.* Zondervan, Grand Rapids, MI, 1994.

[24] MOUNCE, W. D. *Basics of Biblical Greek Grammar.* Fortress, Philadelphia, 2003.

[25] PALMER, M., KINGSBURY, P., AND GILDEA, D. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics 31*, 1 (2005), 71–106.

[26] PERRAULT, C. R. A plan-based analysis of indirect speech acts. *American Journal of Computational Linguistics 6*, 3-4 (1980), 167–182.

[27] PORTER, S. E., REED, J. T., AND O'DONNELL, M. B. *Fundamentals of New Testament Greek.* Eerdmans, Grand Rapids, MI, 2010.

[28] QUARTZ HILL SCHOOL OF THEOLOGY. Ugarit and the bible, 2015.

[29] REITER, E., AND BELZ, A. An Investigation into the Validity of Some Metrics for Automatically Evaluating Natural Language Generation Systems. *Computational Linguistics 35*, 4 (2009), 529–58.

[30] REITER, E., AND DALE, R. *Building Natural Language Generation Systems.* Cambridge University Press, Cambridge, U.K., 2000.

[31] REITER, E., MELLISH, C., AND LEVINE, J. Automatic generation of technical documentation. *Applied Artificial Intelligence 9*, 3 (1995), 259–287.

[32] RITCHIE, G. Computational mechanisms for pun generation. In *Proceedings of the 10th European Natural Language Generation Workshop* (Aberdeen, 2005), pp. 125–132.

[33] RUSSELL, S. J., AND NORVIG, P. *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, Englewood Cliffs, NJ, 2009.

[34] TRENCHARD, W. C. *Complete Vocabulary Guide to the Greek New Testament.* Zondervan, Grand Rapids, MI, 1998.

[35] VAUQUOIS, B. A survey of formal grammars and algorithms for recognition and transformation in machine translation. *Proceedings of the IFIP Congress-6* (1968), 254–260.

[36] WALLACE, D. B. *Greek Grammar beyond the Basics: An Exegetical Syntax of the New Testament with Scripture, Subject, and Greek Word Indexes.* Zondervan, Grand Rapids, MI, 1996.

[37] WALLACE, D. B. *The Basics of New Testament Greek Syntax: An Intermediate Greek Grammar.* Zondervan, Grand Rapids, MI, 2000.

[38] WOODS, W. A. *Semantics for a Question-Answering System.* Harvard University, 1967.

[39] WU, A., AND TAN, R. *Cascadia Syntax Graphs of the New Testament: SBL Edition.* Lexham Press, 2010.

[40] YOUNG, R. A. *Intermediate New Testament Greek: A Linguistic and Exegetical Approach.* Broadman & Holman, Nashville, TN, 1994.